Static Resource Models of Instruction Sets

Q. Zhao
Dept. of Elec. Eng.
Eindhoven University of Technology
NL-5600 MB Eindhoven
The Netherlands
q.zhao@tue.nl

T. Basten
Dept. of Elec. Eng.
Eindhoven University of Technology
NL-5600 MB Eindhoven
The Netherlands
a.a.basten@tue.nl

B. Mesman*
Philips Research Laboratories
Prof. Holstlaan 4
NL-5656 AA Eindhoven
The Netherlands
bart.mesman@philips.com

ABSTRACT

Due to an increasing need for flexibility, embedded systems embody more and more programmable processors as their core components. Because of silicon area and power considerations, the corresponding instruction sets are often highly encoded to minimize code size for given performance requirements. This has hampered the development of robust optimizing compilers because the resulting irregular instruction set architectures are far from convenient compiler targets. Among others, they introduce a strong phase coupling between the tasks of instruction selection and scheduling. Traditional methods perform these tasks in different phases, thereby yielding inferior schedules. In this paper, we present an approach that reduces the need for explicit instruction selection by transferring constraints implied by the instruction set to static resource constraints. All resulting schedules are then guaranteed to correspond to a valid implementation with available instructions. We demonstrate a practical way to identify and construct a static resource model from a given instruction set. Experimental results show the efficacy of our approach.

Keywords

code generation, high-level synthesis, instruction set constraints

1. INTRODUCTION

The combined issues of performance requirements for meeting real-time constraints on the one hand, and code size requirements on the other hand, have led embedded DSP vendors to focus on the "expressive power" of the associated instruction sets. The resulting instruction sets are highly encoded and have an irregular structure, making them inconvenient compiler targets. Overheads in cycle time and code size have been reported in the order of 800% [10] when compared to manually written assembly. As a result, these embedded DSP processors are most often programmed in assembly, which is painstaking, labour intensive, and requires expert knowledge of the processor architecture and instruction set. Furthermore, the resulting code is not portable to other platforms, difficult to debug, maintain and update, etc.

The task of instruction selection is to *cover* a data flow graph (DFG), like the one depicted in Figure 1, with processor instructions that implement the individual operations in the DFG. The main issue introduced by an irregular instruction set is the issue of *phase coupling*: On the one hand, if instruction selection is performed prior to scheduling, the optimal schedule can easily be eliminated as a result of the choices made during instruction selection. On the other hand, if scheduling is performed first, the available instructions may not be able to implement the schedule. Traditional methods perform these tasks in different phases, thereby yielding inferior schedules. This point is illustrated in Figure 1. The DFG on the left hand side has been covered with machine instructions. The associated optimal schedule (6 clock cycles) for this selection of instructions is given in the right hand side of Figure 1. The imposed instruction selection is rather unfortunate, however, because Figure 2 depicts a valid schedule requiring only 5 clock cycles.

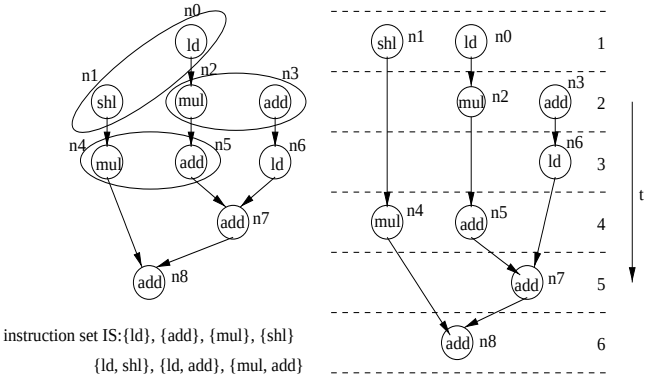

Figure 1: *Instruction selection prior to scheduling may yield inferior results*

In this paper, we present a new approach that eliminates the need for explicit instruction selection by transferring the

*See Section 7 for additional authors.

ISSS'01, October 1-3, 2001, Montréal, Québec, Canada.

constraints from the instruction set to static resource constraints. All resulting schedules are then guaranteed to correspond to a valid implementation with instructions. This is illustrated in Figure 2. The instruction set IS of Figure 1 has been augumented with virtual resources $\{ld, mul\}$, $\{mul, shl\}$ and $\{shl, add\}$, which are abbreviated as LM, MS and SA. Each of these virtual resources has one instance. Each operation uses all the virtual resources that it is contained in, e.g., operation mul uses virtual resources LM and MS. For this so called *static resource model* (SRM), a straightforward list scheduler generates the schedule on the right hand side of Figure 2. The reader can verify that the operations at each clock cycle can be implemented with instructions from the original instruction set IS. This example demonstrates that better schedules can be obtained using a static resource model thereby reducing the need for explicit instruction selection.

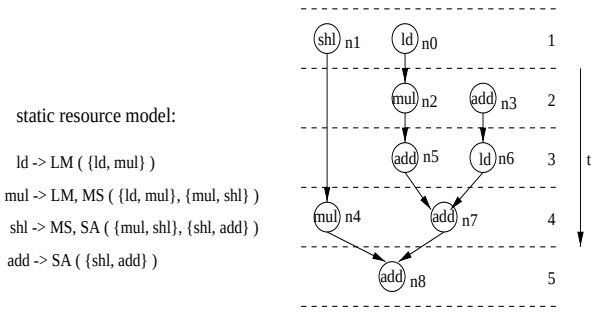

Figure 2: *With a static resource model optimal results can be obtained*

Much work has been done on exploiting instruction selection. Most compilers use template pattern bases and tree-covering techniques to partition the instruction selection problem in a stepwise fashion [1, 5, 8]. Such methods often split the DFG into expression trees and perform tree-covering on each tree separately, demonstrated by the example in Figure 1. In *Chess* [13], the target processor is described in the nML language [3] and is translated into an Instruction-Set-Graph (ISG), which models connectivity, encoding restrictions and structural hazards. Code selection covers the control-data flow graph with partial instructions (bundles) by searching valid paths in the ISG. Such a method adds a complex step to the compiler chain and eliminates potentially interesting solutions from the schedule and register binding search spaces. In [4, 14], code generation tasks are performed by imposing constraints and the complete solution space is explored while all the constraints are considered. This approach can only deal with small applications and a restricted set of architectures.

Another approach to tackle the code generation problem is to exploit the instruction set constraints statically. Eisenbeis [2] starts from the placement conflicts by using reservation tables for VLIW architectures, such as the Trimedia processor. Timmer et al. [12] discusses the modeling of instruction set constraints together with resource constraints for instruction scheduling, but the assumption that all the instruction set constraints can be transferred to resource constraints targets mostly instruction sets with a structure associated with so called Issue-Slot machines. The virtue of these orthogonal instruction sets, like VLIW instruction sets [11], is that they allow to a large extent the modeling of the instruction set constraints by means of a model like [2], [12], or a static resource model, as proposed in this paper. These models offer the scheduler more opportunity to satisfy the timing and resource constraints than with the use of an explicit instruction selection step. The scheduler rather than the instruction selector is considered the designated place for handling these constraints. The disadvantage of the orthogonal processors (especially VLIW processors) is the inherent problem of code size due to their associated instruction word widths. The focus of this paper is on replacing the instruction set constraints of highly encoded instruction sets with a static resource model, allowing the use of code size efficient processors in combination with efficient compilation tools. In this paper we do not consider the strong phase coupling between scheduling and register binding; for that, we use the method described in [9].

This paper is organized as follows. Section 2 details the problem statement and approach. Section 3 addresses the identification and construction of an SRM of an instruction set. In Section 4, some rules are provided for the minimization of SRMs. In Section 5, experimental results are given for benchmarks as well as real applications. Conclusions and future work are discussed in Section 6.

2. PROBLEM STATEMENT AND APPROACH

A DSP algorithm can be expressed as a data flow graph, which describes the primitive operations performed in an algorithm and the dependencies between those operations.

Definition 1. A *data flow graph* (DFG) is a tuple (V, E), where V is the set of vertices (operations) and $E \subseteq V \times V$ is the set of precedence edges.

In a processor architecture, a functional resource can be used in different ways. E.g., a functional resource ALU can execute an operation *add* or a *subtract*, etc. For reasons of complexity, we do not wish to enumerate all possible uses of a functional resource in an instruction set. Therefore, we consider the collection of these uses of a functional resource and associate with it an *operation type*. E.g., on the functional resource ALU, operations *add* and *subtract* can be executed, which are associated with the operation type alu. We denote the set of operation types with T. An *instruction* is now defined as a combination of operation types that can be executed in a single clock cycle.

An operation type can appear multiple times in an instruction (multiple alus can be present in the data path). For an operation type op in an instruction I, we denote this number by $I(op)$. If for two instructions I_0 and I_1, $I_0(op)$ is always at most equal to $I_1(op)$ for each operation type op, we say that instruction I_0 is *contained* in instruction I_1. In this paper we consider instruction sets IS where for each instruction all contained instructions are also in IS. We call these instruction sets *prefix closed*.

The code generation problem is to find a schedule of a DFG, i.e., to determine a start time $s(v)$ for each operation $v \in V$, that satisfies precedence constraints and architectural constraints. These architectural constraints can be modeled either as an instruction set or by introducing functional resources and associating a certain resource usage with each operation. The corresponding resource constraints are *static* in the sense that they only provide a fixed upper limit and any usage within the limit is valid.

Definition 2. A *Static Resource Model* (SRM) is a model generating static resource constraints that defines three aspects:

−a set of resources R,

−a mapping that associates each operation type with the resources in R that it needs, and

−the number of instances of each resource $r \in R$, denoted by $\#r$.

Our approach is motivated by the observation that both resource constraints and instruction set constraints can be expressed as inequalities. For example, if an architecture contains two ALUs and each ALU can be used as an adder or a subtractor, then the resource constraints can be expressed as the following inequality: $N(ALU) \leq 2$, which is equal to: $N(A) + N(S) \leq 2$, where $N(A)$ and $N(S)$ denotes the number of adders and subtractors in use. Any schedule satisfying at any time the above inequality indicates a valid resource usage. Similarly, if an instruction set contains instructions $\{add, add\}$, $\{add, sub\}$ and $\{sub, sub\}$, the operation type usage can also be expressed as an inequality: $N(add) + N(sub) \leq 2$, assuming any subinstruction is also a valid instruction.

Problem Statement: The general problem can be defined as mapping a given instruction set to an SRM such that for any schedule for a DFG satisfying the functional resource constraints posed by the SRM, a corresponding valid instruction selection exists. We say that the instruction set *has* an SRM. This problem can be separated into two subproblems.

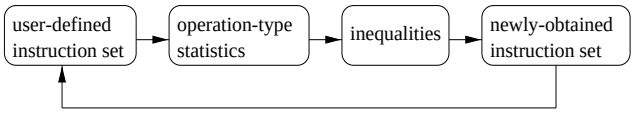

Figure 3: *Overview of the approach*

Problem 1. Given an instruction set, determine whether it has an SRM.

We propose a solution strategy based on expressing the instruction set constraints as inequalities, like $N(add) + N(sub) \leq 2$ in the example above. These inequalities follow from the *operation-type statistics*, as explained in the next section. We then search for an SRM that implies the same set of inequalities. The equivalence of the instruction set constraints and the SRM is verified by comparing the original instruction set to the instruction set that is implied by the SRM. The procedure is illustrated in Figure 3.

Because the run-time performance of a scheduler depends heavily on the complexity of the SRM, we also consider the following problem.

Problem 2. For an instruction set having an SRM, construct the SRM with minimal resources.

The resources can be obtained directly from the inequalities when deriving an SRM. Section 4 provides three rules for reducing the number of resources by analyzing potential redundancy in the generated inequalities.

3. IDENTIFICATION AND CONSTRUCTION OF AN SRM

In this section, we present a method based on inequalities for verifying whether an instruction set IS has an equivalent SRM. If it has, the method also gives this SRM.

3.1 Deriving inequalities

Consider the instruction set of Figure 4(a). The desired inequalities are derived from the *operation-type statistics* of the instruction set, depicted in the table in Figure 4(b). In the table, rows correspond to the individual instructions in Figure 4(a), and columns correspond to (combinations) of operation types. The numbers in the table indicate how many times an operation type is present in an instruction. For example, the operation type *add* occurs twice in instruction (1), and the operation types *shift* and *mul* together occur three times in instruction (3). By looking at the largest frequency within each column, the inequalities in Figure 4(c) are derived. Each inequality corresponds to one column.

(1) {add,add,mul,shift}
(2) {add,add,shift,shift}
(3) {add,mul,mul,shift}
(4) {add,mul,shift,shift}

	add	mul	shift	add mul	add shift	shift mul	add shift mul
(1)	2	1	1	3	3	2	4
(2)	2	0	2	2	4	2	4
(3)	1	2	1	3	2	3	4
(4)	1	1	2	2	3	3	4

(a) user-defined instruction set

(b) operation-type statistics

(1) N(a) <= 2
(2) N(m) <= 2
(3) N(s) <= 2
(4) N(a)+N(m) <= 3
(5) N(a)+N(s) <= 4
(6) N(m)+N(s) <= 3
(7) N(a)+N(m)+N(s) <= 4

(c) inequalities

#A = 2, #M = 2, #S = 2,
#AM = 3, #MS = 3, #AMS = 4

add -> A, AM, AMS
mul -> M, AM, MS, AMS
shift -> S, MS, AMS

(d) static resource model

Figure 4: *Example of an instruction set with an SRM*

In general, given some instruction set IS, we have one inequality for each set of operation types $S = \{op_1, \cdots, op_n\}$:

$$N(op_1) + \cdots + N(op_n) \leq \underset{I \in IS}{\mathrm{MAX}}\Big(\sum_{op \in I \cap S} I(op)\Big) \qquad (1)$$

where $N(op_i)$ is the number of times an operation type appears in an instruction.

The instruction set constraints have now been replaced by a set of inequalities, denoted by $InEq(IS)$. This set often contains redundancy; in the example of Figure 4, inequality (5) can be removed because it is implied by inequality (7). Removing inequalities is advantageous because the complexity of the derived SRM is determined by the number of inequalities in $InEq(IS)$.

3.2 Deriving an SRM

The desired SRM can be derived from the inequalities obtained from the operation-type statistics. Our choice to consider prefix closed instruction sets is motivated by the following corollary, which follows from the containment relationship of instructions.

Corollary. If instruction I_1 contains instruction I_0, and I_1 satisfies $InEq(IS)$, then I_0 also satisfies $InEq(IS)$.

This corollary implies that we can construct an SRM by limiting our observation to the "largest" instructions that are not contained in any other instruction.

Theorem. If S is the set of all the instructions satisfying $InEq(IS)$ and S is equal to the original IS, then IS has an SRM.

We prove this theorem by giving the following SRM:

- R equals the *sets* of operation types,
- each operation type needs all the resources that it is contained in, and
- the number of instances of a resource equals the bound in the corresponding inequality defined by (1).

For example, inequality (4) in Figure 4(c) translates to a virtual resource $\{add, mul\}$, which is abbreviated as AM, with three instances available. For reasons of convenience, we represent a virtual resource by combining the first letters of all those operation types which compose the virtual resource. This will be used throughout the paper. Thus AM is used by operation types add and mul, as indicated in Figure 4(d).

In order to close the loop in Figure 3 and to prove the above theorem, we need to verify that the given SRM models the original instruction set. We do this by implicitly enumerating the instructions that are valid given the SRM. If this newly-obtained instruction set is the same as the original instruction set, then we may conclude that the original instruction set has an SRM. It is not difficult to see that a schedule of a DFG constrained via the above SRM satisfies all the instruction set constraints.

Our method does not always give an SRM. For example, if we omit instruction (4) in the above example, we obtain the same seven inequalities as before and thus also the same SRM. However, this SRM is not an SRM for instruction set IS = {(1), (2), (3)} because it allows instruction (4) which is not an element of IS.

Note that for certain important classes of instruction sets, it is always possible to derive an SRM using our method. One such class are the VLIW instruction sets, which serve as the interface for VLIW architectures that are used quite often in media processing. Our method is a generalization of the methods presented in [2] and [12].

4. MINIMIZATION OF AN SRM

As we have seen in the example of Figure 4, sometimes some of the inequalities derived via our method are redundant. If we can remove an inequality, we can also remove the corresponding resource from the derived SRM. However, the general problem of minimizing a set of inequalities is intractable. Thus, we resort to heuristics. We summarize three rules for minimizing an SRM. For simplicity, the rules are presented only for two operation types op_i and op_j. Assume that the usage constraints for virtual resources $\{op_i\}$, $\{op_j\}$ and $\{op_i, op_j\}$ are given as follows: $N(\{op_i\}) = n_i$, $N(\{op_j\}) = n_j$ and $N(\{op_i, op_j\}) = n_{ij}$.

Rule 1. If $n_i = n_j = n_{ij}$, then operation types op_i and op_j are *totally overlapping*, which means that they share the same resource instances at any time for any usage. Virtual resources $\{op_i\}$ and $\{op_j\}$ can be removed from the SRM, because the constraint on $\{op_i, op_j\}$ makes the constraints on these two resources redundant.

Rule 2. If $n_i < n_j$ and $n_j = n_{ij}$ then operation type op_j *fully covers* op_i, or operation type op_i is *fully covered* by op_j. Each execution of op_i will occupy the resource instances for executing op_j, but not all the usage of op_j will be restricted by op_i. In this case, resource $\{op_j\}$ is redundant since the constraint imposed by resource $\{op_i, op_j\}$ already enforces the constraint imposed by $\{op_j\}$.

Rule 3. If $n_i + n_j = n_{ij}$, then operation types op_i and op_j are *non-overlapping*, which means they don't share any resource instances at all. In this case, resource $\{op_i, op_j\}$ is redundant. For instance, in Figure 4, resources $\{add\}$ and $\{shift\}$ are non-overlapping, which indicates that inequality (5) can be removed.

Note that if $n_i + n_j > n_{ij}$, then operation types op_i and op_j are *partly overlapping*. Part but not all of the resource instances which can execute op_i can also be used for op_j, and vice versa. In this case, we cannot remove any resources.

Reconsider the example in Figures 1 and 2. The minimized SRM of Figure 2 has been obtained by repeatedly applying rules 1 and 3.

5. EXPERIMENTAL RESULTS

The first example shows that our method can also deal with very flexible designs using complex instructions. The task to be scheduled is a slightly adapted example taken from [7]:

$e = m_1 * m_2 + m_3 * m_4 + (m_5 * m_6 + m_7 * m_8) * m_9$

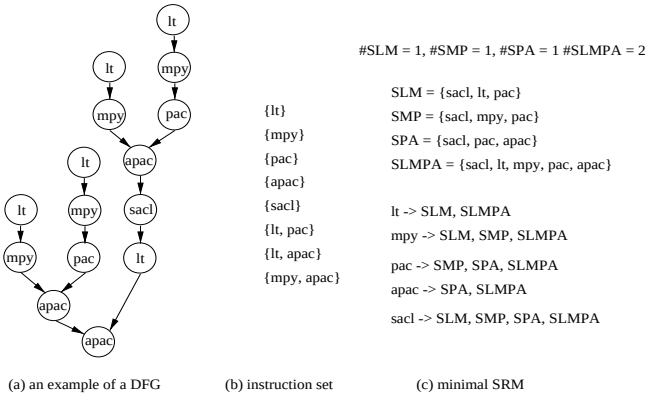

Figure 5: *Complex instructions and the SRM*

Complex instructions for the TMS320C25 processor are shown in Figure 5(b). They indicate potential parallelism to be exploited. This instruction set can be proven to have an SRM and this SRM can be minimized by applying the rules in Section 4. Thus the minimized SRM is given in Figure 5(c). Scheduling the DFG in Figure 5 with this SRM gives us a latency of 12, which equals the optimal result.

One of the merits of the SRM approach is that by performing instruction scheduling for applications with different SRMs, one can directly see from the results how restrictive an IS is. Experiments below perform the scheduling and register binding for benchmarks with two instruction sets. IS1 is the instruction set in Figure 4, IS2 is the set $\{\{add, add, shift, shift\}, \{mul, mul, shift, shift\}, \{add, mul, shift, shift\}\}$. Assume that the hardware resources are ALU, MUL and $SHIFT$, each with two instances and one cycle delay. Table 1 lists the results for 'fdct', 'idct', 'ar' for AR filter and 'wdf' for fifth-order digital elliptical wave filter. The second and third columns show the latency and register requirements (assuming one register file for all

the variables) with given functional resources, the fourth and fifth give latency and registe requirements for IS1 and the sixth and seventh for IS2. #in is the number of minimized inequalities.

Table 1: Scheduling and register binding results

	FS		IS1		IS2	
	L	RF	L_{SRM}	RF_{SRM}	L_{SRM}	RF_{SRM}
fdct	13	12	16	12	21	14
idct	14	11	15	11	20	13
ar	10	6	11	7	14	8
wdf	16	8	16	8	19	11
#in	3		6		4	

From this table we can see that the latency and register requirements with only the functional resource restrictions are minimal for all the examples. This is because we assume that all the functional resources can be exploited maximally. Instruction set constraints usually increase the latency and register requirements. Although using the same resouces, IS2 is obviously more restrictive than IS1, since all latencies are increased. Register requirements are also increased because when some operations are postponed in order to meet the instruction set constraints, the values they consume have to be stored in registers for a longer time, which increases the register pressure.

Table 2: TMS320C62x instruction set

	L	S	D	M
Integer Adder	add	add	add	
	sub	sub	sub	
	mov	mov	mov	
	cmpgt			
	cmplt			
Logic	and	and	and	
	or	or	or	
	not	not	not	
Shift		shl		
		shr		
			ld	
			st	
Multiplier				mpy

VLIW instruction sets have an SRM because of their orthogonal instruction format, such as the TMS320C62x architecture. The C62x has two identical data paths with four functional units each. Each data path has 16 32-bit registers. Table 2 shows part of the instructions for one group of the data path supported by TMS320C62x.

Before computing operation-type statistics, some optimization can be performed to operation types which have exactly the same usage in the instruction set. For example, *add*, *sub* and *mov* can be collected in one group A, *comgt* and *comlt* in group C, *and*, *or* and *not* in group L, *shl* and *shr* in group S, *ld* and *st* in group D. Now virtual resources can be created and they can be minimized by applying the rules in Section 4. The resulting SRM can be summerized as follows: $\#M = 1, \#C = 1, \#S = 1, \#D = 1, \#CLS = 2, \#ACLSD = 3$. Table 3 shows the scheduling and register binding results for GSM speech-coding algorithms. 'Convolution' and 'viterbi' are the half-rate convolutional encoder and one_viterbi_step procedure for the viterbi decoder in channel codec. 'weight', 'inverse' and 'quant' are the weighting_filter, APCM_inverse_quantization and APCM_quantization algorithms for regular pulse exitation encoding in speech codec. SRMs can also be applied to software pipelining, since it only provides a static boundary without modifying the scheduling and register binding algorithm themselves. In Table 3, L and RF are the latency and register requirements with instruction-set constraints within one iteration. II is the minimal initiation interval when software pipelining is applied. L' and RF' are the latency and register requirements under the minimal initiation interval restrictions. This is not performed for 'quant' because it costs too much time to obtain the minimal initiation interval.

Table 3: Scheduling and register binding results for GSM speech-coding algorithms

	L	RF	II	L'	RF'
convol	5	8	5	5	8
viterbi	21	10	18	33	10
weight	18	7	11	20	7
invers	7	4	2	8	5
quant	89	16	-	-	-

6. CONCLUSIONS

In this paper, we propose a method for code generation for highly encoded instruction set processors. The purpose of the method is to deal with the strong phase coupling between instruction selection and scheduling, caused by the instruction set. We replace the instruction set with a set of virtual resources that implement the individual operations contained in the instruction set. This reduces the need for explicit instruction selection, because any schedule satisfying the virtual resource constraints can be implemented by the instruction set. With this static resource model, the scheduler has the opportunity to generate better schedules in terms of timing and register requirements. Furthermore, software pipelining can be applied more effectively to exploit the available ILP.

Our experiments demonstrate that a static resource model of an instruction set offers the possibility to measure the effectiveness of an instruction set with respect to exploiting the processor resources. This enables instruction set designers to make tradeoffs between the performance and the code size associated with an instruction set.

We plan to extend our method to support more versatile instruction sets. In addition, the heuristics to remove the redundancies from an SRM need more investigation. We also intend to study special features of instruction sets used in media processors, such as SIMD instructions [6].

7. ADDITIONAL AUTHORS

C.A.J. van Eijk (Magma Design Automation BV, email: koen@Magma-DA.com) and J.A.G. Jess (Eindhoven University of Technology, email: J.A.G.Jess@ele.tue.nl).

8. REFERENCES

[1] G. Araujo, S. Marlik, and M. Lee. Using Register-Transfer Paths in Code Generation for Heterogeneous Memory-Register Architectures. In *33rd Design Automation Conference Proceedings*, pages 591–596, June 1996.

[2] C. Eisenbeis, Z. Chamski, and E. Rohou. Flexible Issue Slot Assignment for VLIW Architectures. In *Proceedings of the 4th Int. Workshop on Software and Compilers for Embedded Systems*, March 1999.

[3] A. Fauth et al. Describing Instruction Set Processors Using nML. In *European Design and Test Conference Proceedings*, pages 503–507, March 1995.

[4] S. Hanono, G. Hadjiyiannis, and S. Devadas. Aviv: A Retargetable Code Generator Using ISDL. In *34th Design Automation Conference Proceedings*, pages 299–302, June 1997.

[5] R. Leupers. *Retargetable Code Generation for Digital Signal Processors*. Kluwer Academic Publishers, Amsterdam, 1997.

[6] R. Leupers. Code Selection for Media Processors with SIMD Instructions. In *Design, Automation and Test in Europe Conference Proceedings*, pages 4–8, March 2000.

[7] R. Leupers and P. Marwedel. Instruction Selection for Embedded DSPs with Complex Instructions. In *Europe Design Automation Conference*, 1996.

[8] C. Liem, T. May, and P. Paulin. Instruction-Set Matching and Selection for DSP and ASIP Code Generation. In *Proceedings of the Int. Symposium on System Synthesis*, 1994.

[9] B. Mesman, C. Alba-Pinto, and C. van Eijk. Efficient Scheduling of DSP Code on Processors with Distributed Register Files. In *Proceedings of the Int. Symposium on System Synthesis*, November 1999.

[10] P. Paulin and C. Liem. Embedded Systems: Tools and Trends. In *European Design and Test Conference Proceedings*, March 1996.

[11] B. Rau and C. Glaeser. Some Scheduling Techniques and An Easily Schedulable Horizontal Architecture for High Performance Scientific Computing. In *Proceedings of the 14th Workshop on Microprogramming*, pages 183–198, 1981.

[12] A. Timmer, M. Strik, J. van Meerbergen, and J. Jess. Conflict Modeling and Instruction Scheduling in Code Generation for In-House DSP Cores. In *32nd Design Automation Conference Proceedings*, pages 593–598, June 1995.

[13] J. van Praet, G. Goossens, D. Lanner, and H. De Man. Instruction Set Definition and Instruction Selection for ASIPs. In *Proceedings of the Int. Symposium on System Synthesis*, 1994.

[14] T. Wilson, G. Grewal, B. Halley, and D. Banerji. An Integrated Approach to Retargetable Code Generation. In *Proceedings of the Int. Symposium on System Synthesis*, 1994.