HAL
open science

Flexible Issue Slot Assignment for VLIW Architectures

Zbigniew Chamski, Christine Eisenbeis, Erven Rohou

To cite this version:

Zbigniew Chamski, Christine Eisenbeis, Erven Rohou. Flexible Issue Slot Assignment for VLIW Architectures. [Research Report] RR-3784, INRIA. 1999. inria-00072876

HAL Id: inria-00072876

https://inria.hal.science/inria-00072876v1

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Flexible Issue Slot Assignment for VLIW Architectures

Zbigniew CHAMSKI , Christine EISENBEIS , Erven ROHOU

N ° 3784

Octobre 1999

THÈME 1A

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--3784--FR+ENG

Flexible Issue Slot Assignment for VLIW Architectures*

Zbigniew CHAMSKI[†] , Christine EISENBEIS[‡] , Erven ROHOU[§]

Thème 1A — Réseaux et systèmes
Projet A3

Rapport de recherche n° 3784 — Octobre 1999 — 12 pages

Abstract:

Programming specialized processors requires solving complex resource constraints related to the underlying architecture. Although one instruction of the Philips TriMedia VLIW processor can issue five parallel operations, each category of operations can only be allocated to a subset of the five available slots. In this report we show how these restrictions can be translated into constraints based on reservation tables. This allows us to directly apply all classical algorithms for code generation and optimization. An important byproduct is that dynamic processes, such as "on the fly" code generation, are made tractable, even though resource constraints are strongly static.

(Résumé : tsvp)

* This work was partially funded by the ESPRIT IV reactive LTR project OCEANS, under contract No.22729

† Philips Research Laboratories, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands, zbigniew.chamski@philips.com

‡ Inria Rocquencourt, 78153 Le Chesnay, France, Christine.Eisenbeis@inria.fr

§ Harvard University, 33 Oxford Street, Cambridge, MA, 02138, USA, erven@eecs.harvard.edu

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : 01 39 63 55 11 - International : +33 1 39 63 55 11
Télécopie : (33) 01 39 63 53 30 - International : +33 1 39 63 53 30

Architectures VLIW: modélisation des contraintes de slots à l'aide des tables de réservation

Résumé : La programmation des processeurs spécialisés oblige souvent à prendre en compte des contraintes complexes dans l'allocation des ressources, liées à l'architecture sous-jacente. Sur le processeur VLIW TriMedia de Philips, une instruction comporte cinq opérations qui ne peuvent être allouées, selon leur type, que sur un sous-ensemble des cinq slots disponibles. Nous montrons dans ce rapport comment traduire ces contraintes en termes de "tables de réservation", modèle qui permet d'appliquer directement tous les algorithmes classiques de génération et d'optimisation de code. Une conséquence de ce résultat est de rendre praticables des processus dynamiques comme la génération de code "au vol", sur ce processeur à programmation statique très contrainte.

1 Introduction

Instruction scheduling for Very Long Instruction Word (VLIW) architectures requires the compiler to provide a conflict-free resource allocation for all operations to be issued within a single instruction word. On early VLIW architectures, each type of operations was assigned to a fixed field in the instruction word, restricting the operation placement to checking the number of candidate operations of each type, then assigning ready-to-be-issued operations to available fields within each operation type.

On the Philips TriMedia TM-1000 architecture, most operations can be issued in several distinct fields, or "issue slots", within the VLIW instruction, leading eventually to placement conflicts when attempting to issue too many operations sharing the same set of valid issue slots.

In this paper, we present a method for representing such placement constraints using the concept of resources and reservation tables. In our approach, conflict-free placement configurations are modelled using replicated virtual resources, and the standard methods for manipulating reservation tables are used to check the feasibility of a slot assignment. We present a formulation of the slot assignment problem which is then mapped to a resource-based model, we demonstrate the equivalence of both representations, and we present the resource-based model of the current TriMedia architecture obtained using our approach. Finally, we outline several immediate applications of the method for the current TriMedia architecture and software.

2 Motivation - Slot Constraints on the TriMedia Architecture

On the TriMedia, each VLIW instruction contains five slots. In each of these five slots, only some specified types of operations can take place. For instance, in slot 1 only operations of type ALU, CONST, SHIFTER, DSPALU or FALU can be inserted. Taking these constraints into consideration is not straightforward, since early slot allocation may preclude insertion of operations that would indeed fit if another choice was made.

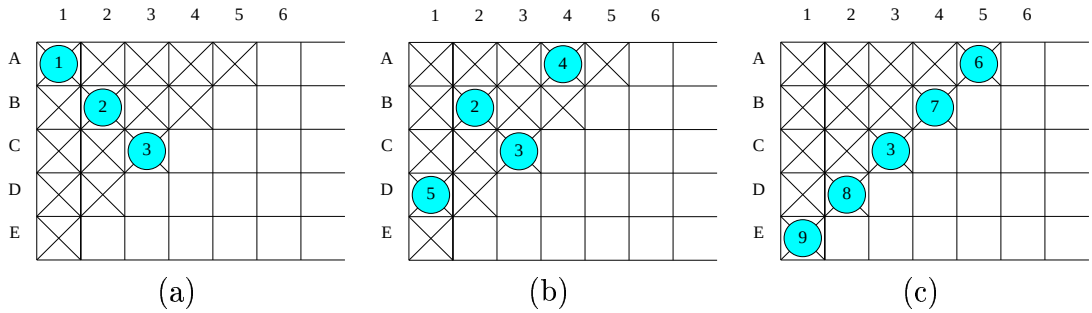

Figure 1: Examples of Constraints in Slot Allocation

Consider for instance the (fictitious) example in Figure 1. In this example, an operation of type A can be inserted in slots 1, 2, 3, 4, or 5, operations of type B in slot 1, 2, 3, or 4, etc. The circled numbers indicate the step at which the given operation was assigned to its current slot. We consider successive scheduling of five operations of respective type A, B, C, D and E. Operation A is scheduled and we assume that slot 1 is chosen. Then operation B is scheduled in slot 2, and operation C in slot 3 (Figure 1a).

Now, let us consider scheduling operation D. D can be placed only in slots 1 or 2, which are already occupied. Without backtracking, operation D can not be scheduled at the same clock cycle. However, it is clear that it could be actually scheduled together with operations A, B and C, provided the slot assignment of these operations is modified. This can be achieved for instance by reconsidering the slot assignment of operation A and putting it in slot 4 (Figure 1b). Now operation E can be put only in slot 1, which is already occupied by operation D. D has to be moved to slot 2 (only possibility). But slot 2 is already occupied by operation B. Hence operation B has to be moved to another slot. The only possibility is slot 4 which in turn is already occupied by operation A. Operation A has to be moved to slot 5 which is fortunately free. Therefore we had to reconsider the scheduling of three already allocated operations to place operation E (Figure 1c).

Another example is given in Figure 2, where a greedy placement strategy of five operations requires the modification of previous slot assignments at each step. The operation of type E is assigned to slot 4 at step 9 only after the slot assignment for operation A has been modified four times, at steps 2, 4, 6, and 8.

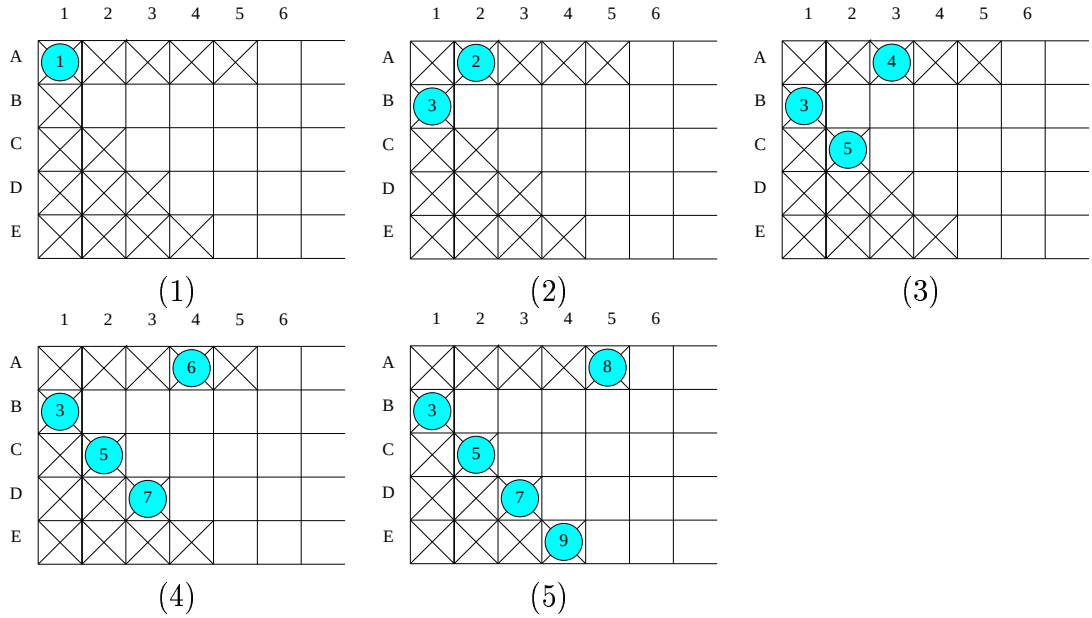

Figure 2: Example of Mandatory Slot Reassignment

Organizing this backtracking process is not an easy task. Actually, any algorithm based on early slot assignment coupled with backtracking may lead to a high computational complexity. In the worst case, all possible combinations of the five operations have to be consid-

ered. The number of these combinations is equal to $n!$ for n slots. It would not be tractable to consider them at each clock cycle.

An alternative approach is to postpone actual allocation of operations to slots, and model slot assignment constraints through the set of valid combinations of operations. Considering whether some operation i may be scheduled at some cycle amounts to answer the question: is there a valid combination that contains i and the set of already accepted operations? This is answered by scanning the list of feasible combinations. In the worst case, an upper bound for the size of this list is $C_t^1 + C_t^2 + C_t^3 + C_t^4 + C_t^5$ items, where t is the number of marks in the table of valid slots for different functional units (Figure 3). In the case of the TriMedia, we can count 21 marks in the table, after identifying types CONST and ALU, and types DSPMUL and IFMUL because they have the same slot constraints. The table contains a total of 27895 entries, computed by enumerating and checking all feasible slot assignments of up to five operation types, including permutations. Searching this table at each tentative of operation scheduling is clearly not tractable.

FU Type \ Slots	1	2	3	4	5
CONST	×	×	×	×	×
ALU	×	×	×	×	×
DMEM				×	×
DMEMSPEC					×
SHIFTER	×	×			
DSPALU	×		×		
DSPMUL		×	×		
BRANCH		×	×	×	
FALU	×			×	
IFMUL		×	×		
FCOMP			×		
FTOUGH		×			

Figure 3: Valid Slots for Functional Units of the TriMedia TM-1000 Processor

The current solution used by Philips and presented in [4] is based on finding a matching in a bipartite graph at each step. The first set is the set of operations under consideration. The second set is the set of the five slots. Edges are drawn between each operation and the set of its valid slots. For dummy operations all slots are valid. A match exists if and only if this combination is valid. This approach has the smallest computational requirements. It is however still adhoc and it may not be extensible to more elaborated resource constraints in the forthcoming architectures, such as operations requiring two adjacent slots in order to be issued.

3 Reservation Tables

Reservation tables are one of the way to model structural hazards in a processor. Reservation tables are two-dimensional tables with time in the horizontal axis and functional units in the vertical axis. Such a table is associated with each operation i. A mark (resp., a number n) is

placed at the intersection of time c and functional unit f if and only if scheduling operation i at time t requires 1 (resp., n) functional unit(s) f at time $t+c$. Then, verifying whether operation i may be scheduled at time t requires verifying the availability of each functional unit f at each cycle $t+c$ for each pair (f, c) marked in the reservation table of operation i.

Let us now describe on examples of Figures 1 and 2 how the reservation tables are formed, then we present a formulation of the general case, and we prove that this formulation is correct. We then give the reservation tables for slot assignment on the TriMedia architecture.

3.1 Examples

In the two examples of Figures 1 and 2, we have the same kind of slot constraints. It is straightforward to imagine how to model these constraints by means of reservation tables. As a matter of fact, for example on Figure 1, we use 5 functional units, corresponding to A, B, C, D, and E. Then operations of type A use one instance of functional unit (A). Operations of type B use one instance of functional unit (A) and one instance of (B), ... and so on. Operations of type E use one instance of each functional unit. Functional unit (A) is replicated five times, functional unit (B) four times, functional unit (C) three times, functional unit (D) two times and functional unit (E) only once.

The slot table of the TriMedia TM-1000 restricted to types SHIFTER, DSPALU and DSPMUL is shown in Figure 4. We need four functional units to model the slot constraints: (SHIFTER) replicated two times, (DSPALU) duplicated two times, (DSPMUL) duplicated two times, and (SHIFTER, DSPALU, DSPMUL) that should be read as "SHIFTER *or* DSPALU *or* DSPMUL", replicated three times. Then operation SHIFTER uses functional unit (SHIFTER) and (SHIFTER, DSPALU, DSPMUL), operation DSPMUL uses functional unit (DSPMUL) and (SHIFTER, DSPALU, DSPMUL), and operation DSPALU uses functional unit (DSPALU) and (SHIFTER, DSPALU, DSPMUL). How this reservation model is derived in the general case is explained in the next Section.

Slots / FU Type	1	2	3	4	5
SHIFTER	X	X			
DSPALU	X		X		
DSPMUL		X	X		

Figure 4: Restricted Slot Table for the TriMedia

3.2 General Case

3.2.1 Formulation

We consider the problem with n slots. $\Sigma = \{s_1, ..., s_n\}$ is the set of slots. For each operation i, we denote as $S(i)$ the set of slots where operation i can be placed. $|S(i)|$ denotes the cardinal of $S(i)$. For each operation i and each slot $s \in \Sigma$ we define the variable $x_{is} \in \{0, 1\}$ where $x_{is} = 1$ if and only if operation i is allocated to slot s. Then for a set I of operations,

the slot assignment constraints can be formulated as follows:

$$\forall s \in \Sigma, \sum_{i \in I} x_{is} \leq 1 \tag{1}$$

$$\forall i \in I, \sum_{s \in \Sigma} x_{is} = 1 \tag{2}$$

$$\forall i \in I, \forall s \in \Sigma, x_{is} = 1 \Rightarrow s \in S(i) \tag{3}$$

Condition 1 means that at most one operation can be allocated in each slot. Condition 2 means that every operation has to be assigned to some slot. The last condition means that operation i can be assigned only to slots in $S(i)$. The set I of operations is a *valid* combination if and only if the system composed of these 3 conditions has a solution.

We replace this system by another one. For each subset J of operations, we consider the following conditions:

$$|J| \leq |\bigcup_{j \in J} S(j)| \tag{4}$$

Then we argue that a set I of operations is a valid combination of operations if and only if each subset J of I satisfies equation 4.

3.2.2 Proof

The fact that system (1,2,3) implies 4 is easy to prove, by noting that

$$|J| = \sum_{j \in J} 1 = \sum_{j \in J} \sum_{s \in \Sigma} x_{js} = \sum_{j \in J} \sum_{s \in S(j)} x_{js} = \sum_{s \in \bigcup_{j \in J} S(j)} \sum_{j \in J} x_{js} \leq \sum_{s \in \bigcup_{j \in J} S(j)} 1 = |\bigcup_{j \in J} S(j)|$$

The reverse implication is somewhat more complex. It is done by recurrence on the cardinal of I.

If $|I| = 1$, then $I = \{i\}$. Condition 4 means that $S(i)$ is non empty, hence it contains some slot to which i can be assigned.

Now, we assume that $|I| = p$ and that implication holds for $1, 2, ..., p-1$. I is composed of operations $i_1, i_2, ..., i_p$. We isolate operation i_p and consider the subset $I' = \{i_1, i_2, ..., i_{p-1}\}$. Then we prove that condition 4 holds for each subset of I' when restricting slots to $\bigcup_{i \in I'} S(i) \setminus \{k\}$, where k is some slot in $S(i_p)$ to be determined.

By contradiction, if the latter property is not true, then for each slot s in $S(i_p)$, we can find a subset J_s of I' such that 4 does not hold for $\bigcup_{i \in J_s} S(i) \setminus \{s\}$. In other words, $|J_s| > |\bigcup_{i \in J_s} S(i) \setminus \{s\}|$. Since we have also that $|J_s| \leq |\bigcup_{i \in J_s} S(i)|$, it follows that $|J_s| = |\bigcup_{i \in J_s} S(i)|$ and $s \in \bigcup_{i \in J_s} S(i)$. We consider the union $J = \bigcup_{s \in S(i_p)} J_s$. By cardinality arguments, it is easy to prove that $|J| = |\bigcup_{i \in J} S(i)|$. More, we have $S(i_p) \subset \bigcup_{i \in J} S(i)$. Hence, $|J \cup \{i_p\}| = |J| + 1 = |\bigcup_{i \in J} S(i)| + 1 = |\bigcup_{i \in J \cup \{i_p\}} S(i)| + 1$. This is in contradiction with hypothesis 4.

We conclude that the assumption was false, and there is some slot k such that recurrence hypothesis is satisfied for I' and slots $\bigcup_{i \in I'} S(i) \setminus \{k\}$. Therefore we can find a lot assignment of operations of I' that lets slot k free for operation i_p. And the proof is completed.

We have replaced the system of constraints by another one that does not need to actually find a solution. This latter condition is now transformed into constraints based on reservation tables.

3.3 Equivalent Reservation Tables

Let us now consider condition 4. To translate this condition into reservation tables, we first need to define "functional units". Instead of considering operations, we consider types of operations (rows in the table of Figure 3). Let T be the set of different types of operations. For the sake of simplicity, we use the same S function for types and operations. $S(t)$ is the set of slots where operations of type t can be assigned. The set F of (artificial) functional units is built by considering all possible combinations of types, $F = \{T' \subset T\}$. For each $f = \{t_1, t_2, ..., t_p\}$, we denote as $|f|$ the number $|\bigcup_{t_i} S(t_i)|$. This is the number of available samples of functional unit f.

The reservation table of operations of type t_i is built as follows: for each functional unit f where t_i appears, then $(f, 0) = 1$, else $(f, 0) = 0$. Reservation table constraints are that no more than $|f|$ samples of f are used at the same cycle. This constraint is exactly equivalent to condition 4.

3.4 Simplification

As expressed above the formulation is still complex, since the number of functional units may be as large as the number of subsets of T (i.e. $2^{|T|}$). We explain now why it is not necessary to consider every possible combination of types. Actually a lot of constraints are implied by other ones.

The main simplifying property is that if $S(i) \subset \bigcup_{j \in J} S(j)$, then the condition $|J| \leq |\bigcup_{j \in J \cup \{i\}} S(j)|$ implies that $|J| \leq |\bigcup_{j \in J} S(j)|$. Hence among functional units, when $f \subset f'$ and $|f| = |f'|$we can remove f.

Another simplifying property is the following: if $f \cap f' = \varnothing$, $|f \cup f'| = |f| + |f'|$, then $f \cup f'$ can be removed.

3.5 Reservation Tables on the TriMedia

By applying the former process to the TriMedia slots constraints as shown on Figure 3 and after simplification, we obtain the 18 functional units shown in Table 1.

This resource set can be further reduced for those classes of applications which do not use all functional unit types, e.g., which do not use cache operations (type DMEMSPEC), or floating-point operations (types FALU, FCOMP, and FTOUGH), or the DSP-oriented functionalities (type DSPALU).

4 Retrieval of Valid Slot Assignments

Once we have formed a valid combination of operations, we still have to find the actual slot assignement, i.e. to solve the system. For instance, once we know that the combination

Functional Unit (f)	Number ($\|f\|$)	Slots
(DMEMSPEC)	1	5
(FTOUGH)	1	2
(FCOMP)	1	3
(FALU)	2	1,4
(DMEM,DMEMSPEC)	2	4,5
(DSPMUL,IFMUL,FCOMP,FTOUGH)	2	2,3
(DSPALU,FCOMP)	2	1,3
(SHIFTER,FTOUGH)	2	1,2
(DMEM,FALU,DMEMSPEC)	3	1,4,5
(DSPALU,FALU,FCOMP)	3	1,3,4
(SHIFTER,FALU,FTOUGH)	3	1,2,4
(SHIFTER,DSPALU,DSPMUL,IFMUL,FCOMP,FTOUGH)	3	1,2,3
(DSPMUL,BRANCH,IFMUL,FCOMP,FTOUGH)	3	2,3,4
(DMEM,DSPALU,FALU,FCOMP,DMEMSPEC)	4	1,3,4,5
(DMEM,SHIFTER,FALU,FTOUGH,DMEMSPEC)	4	1,2,4,5
(DMEM,DSPMUL,BRANCH,IFMUL,FCOMP,FTOUGH,DMEMSPEC)	4	2,3,4,5
(SHIFTER,DSPALU,DSPMUL,BRANCH,FALU,IFMUL,FCOMP,FTOUGH)	4	1,2,3,4
(ALL)	5	1,2,3,4,5

Table 1: The 18 virtual resources modeling the TriMedia

(SHIFTER, DSPALU, DSPMUL) is valid, we need to find out that it can be issued using slots 1, 3, and 2, respectively, leading to the VLIW instruction pattern (SHIFTER, DSPMUL, DSPALU, NOP, NOP).

It is important to note that the two actions - ensuring the validity of a combination of operations, and finding a slot assignment - can be fully separated, and that the search for a slot assignment corresponding to a given combination can be postponed until the last moment - the validity of the combination guarantees that such an assignment exists.

We propose two methods to find a valid slot assignment for a given combination of operations, the former being already implemented.

4.1 First Method - Table of Solutions

The first method is to use a table of possible combinations and corresponding slots allocation. This is a direct way to handle the problem. The generated table, after merging equivalent operation categories, has 674 entries, corresponding to the list of all feasible combinations of slot assignments, sorted in ascending lexicographical order of operation types and with duplicate entries (corresponding to permutations of operations within the VLIW instruction) removed.

This table can be implemented as a 5-dimensional array to make the retrieval of slot assignments fast and to simplify the coding of retrieval routine, but this would result in a waste of memory space. Therefore, a fast search method using an associative lookup table

has been implemented, exploiting the fact that the key field in each entry (the vector of operation types) can be sorted using a total ordering relation, allowing a dichotomy search.

4.2 Second Method - Fast Solution Computing

There is another alternative consisting of computing directly the solution from the combination. We explain this on an example, but this method can be generalized. It is based on a boolean computation.

A SHIFTER operation can be put in slots 1 or 2. We write this as:

$$\text{SHIFTER} = \text{SHIFTER}_1 \oplus \text{SHIFTER}_2$$

We write the same for operations DSPALU and DSPMUL:

$$\text{DSPALU} = \text{DSPALU}_1 \oplus \text{DSPALU}_3$$

$$\text{DSPMUL} = \text{DSPMUL}_2 \oplus \text{DSPMUL}_3$$

Now we have to find a solution for the combination made of SHIFTER *and* DSPALU *and* DSPMUL. This is written as: $\text{SHIFTER} \otimes \text{DSPALU} \otimes \text{DSPMUL}$. Slot conflicts can also be written as: $t_s \otimes t'_s = 0$ for two different operations of type t and t' and a same slot s.

By developing the expression, distributing and applying the slot conflict equation, we get:

$$\begin{aligned}
\text{SHIFTER} \otimes \text{DSPALU} \otimes \text{DSPMUL} &= (\text{SHIFTER}_1 \oplus \text{SHIFTER}_2) \otimes (\text{DSPALU}_1 \oplus \text{DSPALU}_3) \\
&\quad \otimes (\text{DSPMUL}_2 \oplus \text{DSPMUL}_3) \\
&= (\text{SHIFTER}_1 \otimes \text{DSPALU}_3 \otimes \text{DSPMUL}_2) \oplus \\
&\quad (\text{SHIFTER}_2 \otimes \text{DSPALU}_1 \otimes \text{DSPMUL}_3)
\end{aligned}$$

This means that there are 2 solutions of slot allocations for the combination (SHIFTER, DSPALU, DSPMUL). The first one is (1,3,2) corresponding the VLIW instruction (SHIFTER, DSPMUL, DSPALU, NOP, NOP). The second one is (2,1,3) corresponding to the VLIW instruction (DSPALU, SHIFTER, DSPMUL, NOP, NOP).

Although this computation may not be as fast as the search in a table of solutions, it is very practical because it offers an elegant way of finding all solutions and can actually serve for building the table, instead of systematically enumerating the possibilities. It could actually also serve for verifying compatibility of operations during the scheduling process. Operations are compatible if and only if their $\otimes$-product is non-zero.

5 Applications

The main advantage of this technique is to integrate a new kind of constraint with a well-known framework. As a result, a wide range of tools developed in the field of instruction

scheduling need only minor modifications to deal with VLIW code. Code schedulers simply need a post-pass that order operations within a long word instruction, and they are guaranteed to find a solution.

SALTO is a retargetable system for assembly language transformation and optimization [6]. It has been developed at IRISA and we have been using it for several studies related to code scheduling and ILP. SALTO automatically parses assembly files and generates an intermediate representations of the program. The user can then use the available functions to manipulate instructions, basic blocks and CFGs, compute data dependences, etc. SALTO is fully retargetable wrt. the architecture thanks to a machine description file that specifies the syntax of the assembly language and hardware details.

Porting the system to the Philips TM-1000 merely consisted in adding the 18 virtual resources to the machine description file. It has then been successfully used for various applications such as superblock scheduling, software pipelining and predication.

In addition to determining the initial slot assignment, the reservation table model makes it possible to interleave new threads of operations with an already scheduled code. This results from the fundamentally *incremental* nature of the reservation mechanism: any reservation for a single operation can be easily added or undone, provided the resource usage pattern of that operation is known. Therefore, it is possible to add new code to an existing application with very limited side effects. This is achieved by placing the additional code in issue slots left empty in the original schedule, provided the operations in the original code are compatible with the operation(s) to be added in the empty slot(s). It should be noted that adding new operation may result in a change of slot assignment for the operations already scheduled, but the new slot assignment will always be valid.

This method can be applied in several areas: code instrumentation (selective or application-specific profiling), on-the-fly code generation (just-in-time compilation), etc. Such approaches are currently being investigated at Philips Research in Eindhoven.

6 Related Work

Several authors have explored the use of finite-state automata to perform resource assignment [1, 5] as an efficient alternative to reservation tables, primarily on the grounds of space and time efficiency in the compiler itself. However, resource allocation techniques based on finite-state automata show serious limitations when incremental deallocation of resources is needed, as in modulo scheduling (software pipelining), and in instrumentation of already scheduled code. Also, implementation issues related to the use of reservation tables have been discussed in [3], the authors concentrating precisely on the efficiency of the compiler implementation.

In [2], the authors explore the minimization of resource counts and of sizes of reservation tables, based on heuristic searches. Our approach provides a constructive method of building the minimal set of resources needed to fully model slot assignment constraints of an architecture.

7 Conclusion

The concept of reservation tables is a very convenient tool for operation scheduling as far as we are concerned with retargetability, as in SALTO. To the best of our knowledge, only adhoc solutions are used in the current TriMedia compiler. This may preclude fast adaptation to next TriMedia architectures. Also, reservation tables allow to account for more complex architectural constraints (like the restricted number of register writes in one cycle, or multi-slot operations) within a unified conceptual and algorithmic framework.

We have presented a model of reservation tables for slots constraints, and we have proven that this approach is correct. The reservation tables and artificial functional units are automatically generated from the slots constraints. This will allow fast retargetability to other architectures based on the same principle. With our method, an entire class of resource allocation constraints caused by conflicting alternative choices (several "1-in-many" rules to be satisfied simultaneously) can be handled in an elegant way, while providing a constructive method to build the set of resources needed to model these constraints.

References

[1] V. Bala and N. Rubin. Efficient instruction scheduling using finite-state automata. In *28th Annual International Symposium on Microarchitecture*, pages 46–56, November 1995.

[2] A. E. Eichenberger and E. S. Davidson. A reduced multipipeline machine description that preserves scheduling constraints. In *ACM SIGPLAN '96 Conf. on Programming Language Design and Implementation*, pages 12–22, May 1996.

[3] J. C. Gyllenhaal, W. W. Hwu, and B. R. Rau. Optimization of machine descriptions for efficient use. In *29th Annual Symp. on Microarchitecture*, pages 349–358, December 1996.

[4] J. Hoogerbrugge and L. Augusteijn. Instruction scheduling for TriMedia. *Journal of Instruction-Level Parallelism*, 1(1), February 1999.

[5] T. Müller. Employing finite automata for resource scheduling. In *26th Annual International Symposium on Microarchitecture*, pages 12–20, December 1993.

[6] E. Rohou, F. Bodin, and A. Seznec. SALTO: System for assembly language transformation and optimization. Technical Report 1032, Irisa, September 1996. Also available at `ftp://ftp.irisa.fr/techreports/1996/PI-1032.ps.gz`.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399