A Linear Complexity Algorithm for the Generation of Multiple Input Single Output Instructions of Variable Size*

Carlo Galuzzi, Koen Bertels, and Stamatis Vassiliadis

Computer Engineering, EEMCS
TU Delft
{C.Galuzzi, K.L.M.Bertels, S.Vassiliadis}@ewi.tudelft.nl

Abstract. The Instruction-Set extension problem has been one of the major topics in the last years and it is the addition of a set of new complex instructions to a given Instruction-Set. This problem in its general formulation requires an exhaustive search of the design space to identify the candidate instructions. This search turns into an exponential complexity of the solution. In this paper we propose an algorithm for the generation of Multiple Input Single Output instructions of variable size which can be directly selected or combined for Instruction-Set extension. Additionally, the algorithm is suitable for inclusion in a design flow for automatic generation of MIMO instructions. The proposed algorithm is not restricted to basic-block level and has linear complexity with the number of processed elements.

1 Introduction

The use of electronic devices has became a routine in our everyday life. Just consider the devices we are using in the daily basis such as mobile phones, digital cameras, electronic protection systems in the cars, etc. This great variety of devices can be implemented using different approaches and technologies. Usually these functionalities are implemented using either *General Purpose Processors* (GPPs), or *Application-Specific Integrated Circuits* (ASICs), or *Application-Specific Instruction-Set Processors* (ASIPs). GPPs can be used in many different applications in contrast to ASICs which are processors designed for a specific application such as the processor in a TV set top box.

Last years, processors with a customizable architecture, also known as *Application-Specific Instruction-Set Processors* (ASIPs), have became more and more popular. ASIPs are situated in between GPPs and ASICs: they have a *partially customizable Instruction Set* and perform only a limited number of tasks so giving a tradeoff between flexibility, performance and cost. Although performance of an ASIP is usually lower than an ASIC, the design time and non-recurring engineering costs (the one-time charge for photomask development, test, prototype

* This work was supported by the European Union in the context of the MORPHEUS project Num. 027342.

S. Vassiliadis et al. (Eds.): SAMOS 2007, LNCS 4599, pp. 283–293, 2007.

tooling, and associated engineering costs) can be amortized with the multiple addressable applications tuning the processor characteristics toward the requirements of the specific application.

Maximizing the performance of the ASIP is crucial. One of the key issues involves the choice of an *optimal* instruction-set for the specific application given. Optimality can refer to power consumption, chip area, code size, cycle count and/or operating frequency. A computable solution is not always feasible due to many subproblems such as design space exploration or combinatorial problems. In those cases heuristics are used to find a *close-to-optimal* solution.

Basically there are two types of Instruction-Set customizations which can be pursued: the first and most radical one is to generate a complete instruction set for the specific applications [1,2,3]. The second and less drastic one extends an existing instruction set with instructions specialized for a given domain [4,5,6,7]. In both cases the goal is to design an instruction set containing the most important operations needed by the application to maximize the performance.

The first step in this process is the identification of the operations that should be implemented in hardware and the ones that will be executed in software. The operations implemented in hardware are implemented as peripheral devices or they can be incorporated in the processor as new instructions and/or special functional units integrated on the processor.

In this paper we present a linear complexity algorithm for the generation of Multiple Input Single Output (MISO) instructions which can directly undergo a selection process for hardware-software partitioning or can be clustered with different policies for the generation of MIMO instructions [7,8]. More specifically, the main contributions of this paper are:

- an overall linear complexity of the proposed algorithm. The generation of complex instructions is a well known NP problem and its solution requires, in the worst case, an exhaustive search of the design space which turns into an exponential complexity of the solution. Our algorithms generate MISO instructions of variable size suitable for inclusion in a design flow for automatic generation of MIMO instructions as the ones proposed in [7,8]. Our approach springs from the notion of MAXMISO introduced by [9] and, in a similar way, it requires linear complexity in the number of processed elements as proven in Section 4.
- the proposed approach is not restricted to basic-block level analysis and can be applied directly to large kernels.

The paper is structured as follows. In Section 2, background information and related works are provided. In Section 3 and 4, the basic definitions and the algorithm for MISO instruction generation are presented. Concluding remarks and an outline of research conducted are given in Section 5.

2 Background and Related Works

The algorithms for Instruction Set Extensions usually select clusters of operations which can be implemented in hardware as single instructions while

providing maximal performance improvement. Basically, there are two types of clusters that can be selected, based on the number of output values: MISO or MIMO. Accordingly, there are two types of algorithms for Instruction Set Extensions that are briefly presented in this section.

Concerning the first category, a representative example is introduced in [9] which addresses the generation of MISO instructions of maximal size, called MAXMISO. The proposed algorithm exhaustively enumerates all MAXMISOs. Its complexity is linear with the number of nodes. The reported performance improvement is of few processor cycles per newly added instruction. The approach presented in [10] targets the generation of general MISO instructions. The exponential number of candidate instructions turns into an exponential complexity of the solution in the general case. As a consequence, heuristic and additional area constraints are introduced to allow an efficient generation. The difference between the complexity of the two approaches is due to the properties of MISOs and MAXMISOs: while the enumeration of the first is similar to the subgraph enumeration problem (which is exponential) the intersection of MAXMISOs is empty and then once a MAXMISO is identified, its nodes are removed from the set of nodes that have to be successively analyzed. In this way the MAXMISOs are enumerated with linear complexity in the number of nodes.

The algorithms included in the second category are more general and provide more significant performance improvement. However, they have exponential complexity. For example, in [5] the identification algorithm detects optimal convex MIMO subgraphs but the computational complexity is exponential. A similar approach described in [11] proposes the enumeration of all the instructions based on the number of inputs, outputs, area and convexity. The selection problem is not addressed. In [6] the authors target the identification of convex clusters of operations under given input and output constraints. The clusters are identified with a ILP based methodology similar to the one proposed in [7]. The main difference is that in [6] the authors iteratively solve ILP problems for each basic block, while in [7] the authors have one global ILP problem for the entire procedure. Additionally, the convexity is addressed differently: in [6], the convexity is verified at each iteration, while in [7] it is guaranteed by construction. Other approaches cluster operations by considering the frequency of execution or the occurrence of specific nodes [4,12] or regularity [13]. Still others impose limitation on the number of operands [14,15,16,17] and use heuristics to generate sets of custom instructions which therefore can not be globally optimal.

In this paper we propose a linear complexity algorithm based on the notion of MAXMISO introduced by [9]. Although the algorithm for the generation of MAXMISOs instructions requires linear complexity in the number of processed elements, it is not always possible to implement MAXMISOs directly in hardware due to a relatively high number of inputs. A way to address this problem is the use of the MAXMISO algorithm for the generation of MISO instructions of reduced size as described in Section 4. Moreover the generated instructions can be directly selected for hardware implementation as well as clustered with different policies for the generation of MIMO instructions [7,8].

3 Theoretical Background

3.1 MISO and MIMO Graphs

In order to formally present the approach previously presented, we first introduce the necessary definitions and the theoretical foundation of our solution. We assume that the input dataflow graph is a DAG $G = (V, E)$, where V is the set of nodes and E is the set of edges. The nodes represent primitive operations, more specifically assembler-like operations, and the edges represent the data dependencies. The nodes can have two inputs at most and their single output can be input to multiple nodes.

Basically, there are two types of subgraphs that can be identified inside a graph: Multiple Input Single Output (MISO) and Multiple Input Multiple Output (MIMO).

Definition 1. *Let $G^* \subseteq G$ be a subgraph of G with $V^* \subseteq V$ set of nodes and $E^* \subseteq E$ set of edges. G^* is a MISO of root $r \in V^*$ provided that $\forall\, v_i \in V^*$ there exists a path[1] $[v_i \rightarrow r]$, and every path $[v_i \rightarrow r]$ is entirely contained in G^*.*

By Definition 1, A MISO is a connected graph. A MIMO, defined as the union of $m \geq 1$ MISOs can be either connected or disconnected. Let G_{MISO} and G_{MIMO} be the sets of subgraphs of G containing all MISOs and MIMOs respectively. An exhaustive enumeration of the MISOs contained in G gives all the necessary building blocks to generate all possible MIMOs. This faces with the exponential order of G_{MISO}, and since $G_{MISO} \subset G_{MIMO}$[2], of G_{MIMO}. A reduction of the number of the building blocks reduces the total number of MIMOs which it is possible to generate. Anyhow, it can *drastically reduces* the overall complexity of the generation process as well. A trade-off between complexity and quality of the solution can be achieved considering MISO graphs with specific properties.

3.2 MAXMISO and SUBMAXMISO

Definition 2. *A MISO $G^*(V^*, E^*) \subset G(V, E)$ is a MAXMISO (MM) if $\forall v_i \in V \backslash V^*$, $G^+(V^* \cup \{v_i\}, E^+)$ is not a MISO.*

It is known from the set-theory that each MISO is either maximal (a MAXMISO) or there exists a maximal element containing it [8,9]. [9] observed that if A, B are two MAXMISOs, then $A \cap B = \emptyset$. This implies that the MAXMISOs contained in a graph can be enumerated with *linear complexity in the number of its nodes* (see. [9,7,8]).

Let $v \in V$ be a node of G and let $\textsc{Lev} : V \rightarrow \mathbb{N}$ be the integer function which associates a level to each node, defined as follows:

- $\textsc{Lev}(v) = 0$, if v is an input node of G;

[1] A path is a sequence of nodes and edges, where the vertices are all distinct.
[2] $G_{MISO} = \{G^* \subset G,\ s.t.\ N_{Out} = 1\} \subset \{G^* \subset G,\ s.t.\ N_{Out} \geq 1\} = G_{MIMO}$.

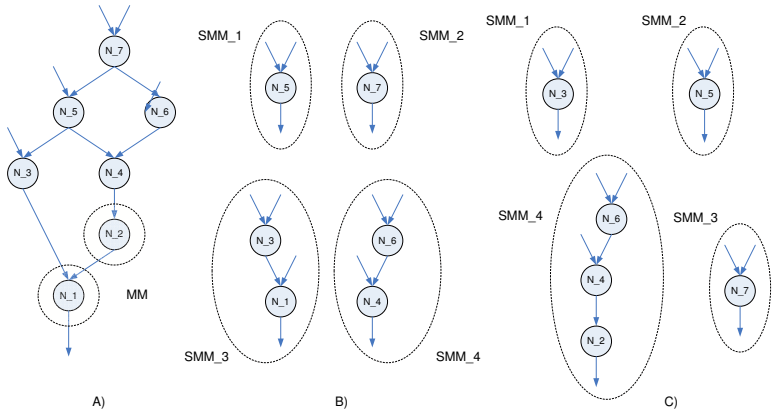

Fig. 1. SMMs of a MAXMISO with different nodes removed: *a*) a MAXMISO MM, *b*) SMMs of $MM \setminus \{N_2\}$, *c*) SMMs of $MM \setminus \{N_1\}$

– $\textsc{Lev}(v) = \alpha > 0$, if there are α nodes on the longest path from v and the level 0 of the input nodes.

Clearly $\textsc{Lev}(\cdot) \in [0, +\infty)$ and the maximum level $d \in \mathbb{N}$ of its nodes is called the **depth** of the graph.

Definition 3. *The level of a MAXMISO $MM_i \in G$ is defined as follows:*

$$\textsc{Lev}(MM_i) = \textsc{Lev}(f(MM_i)). \tag{1}$$

where $f : G \to \hat{G}$ is the collapsing function, the function which collapses the MAXMISOs of G in nodes of the graph $\hat{G}$ (see [8]).

Let's consider a $MAXMISO$ MM_i. Each node $v_j \in MM_i$ belongs to level $\textsc{Lev}(v_j)$. Let $\overline{v} \in MM_i$, with $0 \neq \textsc{Lev}(\overline{v}) \leq d$. If we apply the MAXMISO algorithm to $MM_i \setminus \{\overline{v}\}$, each MAXMISO identified in the graph is called a SUBMAXMISO (SMM) of $MM_i \setminus \{\overline{v}\}$ (or, shortly, of MM_i). Clearly the set of the SMMs tightly depends on the choice of $\overline{v}$ (see Figure 1). For example $\overline{v}$ can be either an exit node (Figure 1c), or an inner node randomly chosen (Figure 1b) or a node with specific properties like area or power consumption below or above a certain threshold previously defined.

The definition of level of a SMM is the obvious extension to SMM of the definition of level of a MAXMISO.

4 The Algorithm for MISO Instruction Generation

In Figure 2 and 3 we present the FIX SMM algorithm and the VARIABLE SMM algorithm respectively. The main difference between the two algorithms is represented by the choice of the node selected for the generation of the SUBMAXMISOs, as outlined in Section 3.2.

Input:= $MM_1, ..., MM_n$
Output:= $SMM_1, ...SMM_k$
—
$SET_1,\ SET_2,\ SET_3 = \emptyset$
for $i = 1..n$ do
{
Choose $\overline{v}_i \in MM_i$
Generate MAXMISO of $MM_i \setminus \{\overline{v}_i\}$
$SET_1 := SET_1 \cup \{MAXMISOs \text{ of } MM_i \setminus \{\overline{v}_i\}\}$
$SET_2 := SET_2 \cup \{\overline{v}_i\}$
$SET_3 := SET_1 \cup SET_2$
}

Fig. 2. FIX SMM Algorithm

a) FIX SMM Algorithm

The main steps of this algorithm are described in Figure 2 and depicted in Figure 4:

a) Given the DAG G of an application, the graph is partitioned in MAXMISOs;
b) For each MAXMISO MM_i we select a node $\overline{v}_i \in MM_i$;
c) $MM_i \setminus \overline{v}_i$ is partitioned in MAXMISOs;
d) Generate the set SET_1 of the SMMs, the set SET_2 of the nodes selected and the set SET_3 union of SET_1 and SET_2.

We have the following property:

Property 1. The complexity of the algorithm is linear in the number of node analyzed.

Proof. This follows from the empty intersection of two MAXMISOs. Let A and B two MAXMISOs, and $\overline{v}_1 \in A$ and $\overline{v}_2 \in B$. Therefore $A \cap B = \emptyset$. This means that:

$$\forall\ MM_i \in A \setminus \overline{v}_1 \text{ and } \forall\ MM_j \in B \setminus \overline{v}_2,\ \ MM_i \cap MM_j = \emptyset. \tag{2}$$

■

b) VARIABLE SMM Algorithm

The main steps of this algorithm are described in Figure 2 and depicted in Figure 4:

a) Given the DAG G of an application, the graph is partitioned in MAXMISOs;
b) For each MAXMISO MM_i a node $\overline{v}_i \in MM_i$ is selected;
c) $MM_i \setminus \overline{v}_i$ is partitioned in MAXMISOs. If the set of SMMs does not satisfy a specific property P_i a different node is selected and the SMMs are regenerated till the property is satisfied.

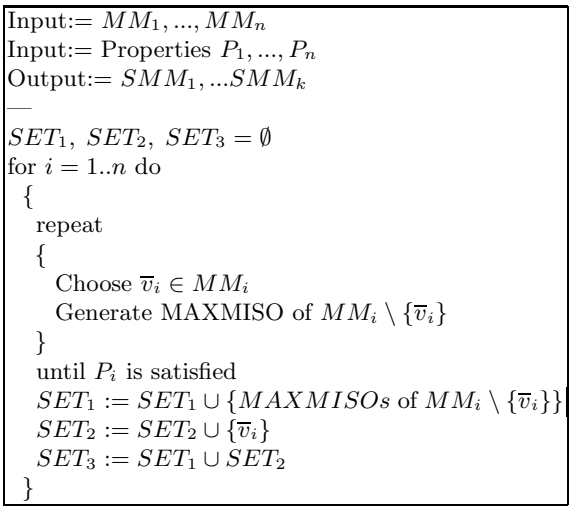

Fig. 3. VARIABLE SMM Algorithm

d) The set SET_1 of the SMMs, the set SET_2 of the nodes selected and the set SET_3 union of SET_1 and SET_2 are generated.

We have the following properties:

Property 2. The complexity of the algorithm is linear in the number of nodes analyzed (as well as for the FIX SMM algorithm as a consequence of the properties of the MAXMISOs).

Property 3. The maximum number of iterations of the algorithm is less than or equal to the order of G[3].

Proof. This follows by the fact that the MAXMISOs are a partition of the graph. For each MAXMISO MM_i is therefore possible to select n_i different nodes. Independently by the value of n_i we have that $\Sigma_i n_i = n$. ■

Remark 1. The algorithm presented in this paper in its two versions, namely FIX SMM, and VARIABLE SMM, is suitable for an iterative process for the generation of MISO instructions of relatively smaller size, when severe input constraints are applied. This can be obtained using as input of the algorithm(s), instead of the set of MAXMISOs $MM_1, ..., MM_n$, the final set SET_1 as described in Section 4.

4.1 Application

In [7,8] we presented two methods for the automatic generation of convex MIMO instructions based on the following result:

[3] The order of a graph $G(V, E)$ with V set of nodes and E set of edges is the order of V.

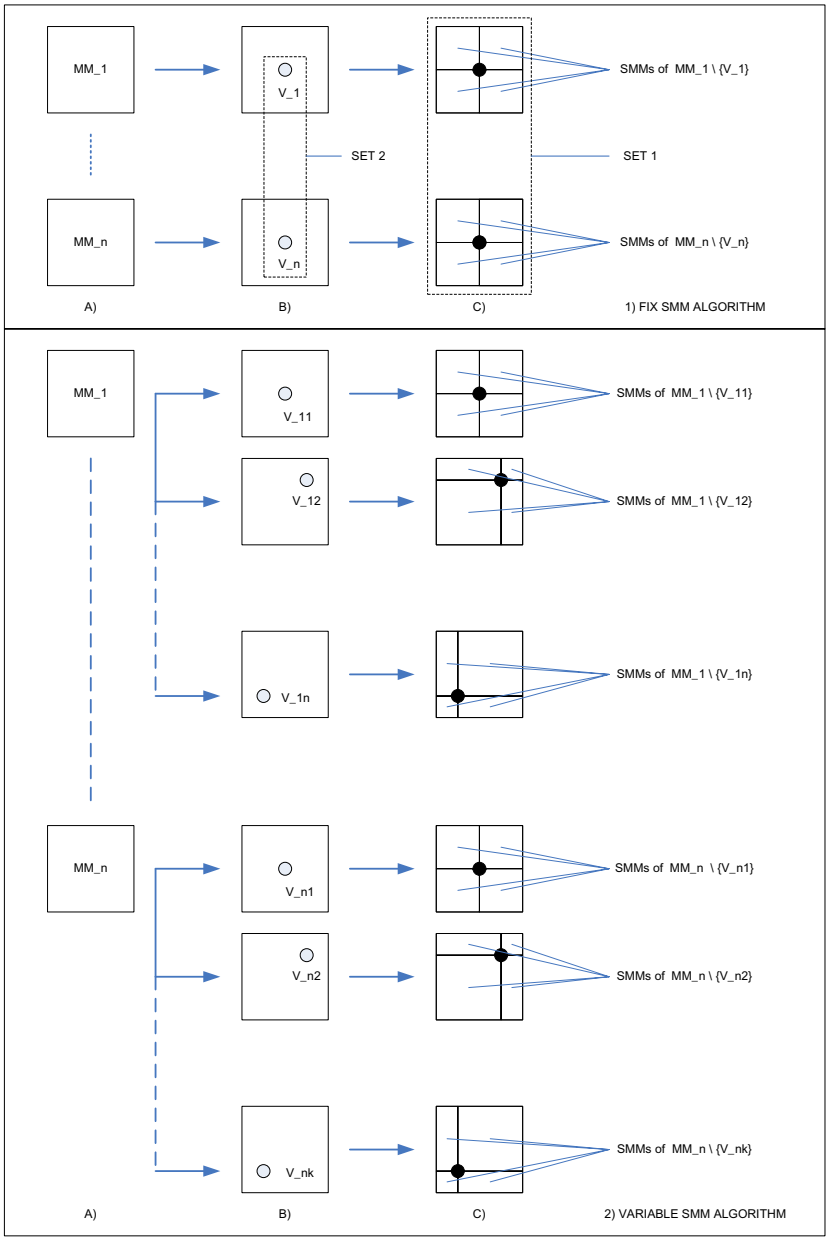

Fig. 4. Description of the main steps required by the algorithms for the generation of the SMMs: 1) FIX SMM algorithm and 2) VARIABLE SMM algorithm. The main steps are A) MM generation, B) selection of the node to remove, and C) SMM generation. (NB In the figure, each MAXMISO is partitioned in 4 SMMs of random size for explanatory reasons. As we have seen in Figure 1 the SMMs depend on the node chosen.)

Theorem 1. *Let G be a DAG and $A, B \subset G$ two MAXMISOs. Let* $\textsc{Lev}(A) \geq \textsc{Lev}(B)$ *be the levels of A and B respectively. Let $C = A \cup B$. If*

$$\textsc{Lev}(A) - \textsc{Lev}(B) \in \{0, 1\} \tag{3}$$

then C is a convex MIMO. Moreover

- *C is disconnected if the difference is 0.*
- *Any combination of MAXMISOs at the same level or at two consecutive levels is a convex MIMO.*

We note that a subgraph $G^\star \subset G$ is convex if there exists no path between two nodes of $G^\star$ which involves a node of $G \backslash G^\star$. Convexity guarantees a proper and feasible scheduling of the new instructions which respects the dependencies.

This theorem can be extended to SMMs.

Corollary 1. *Any combination of SMMs at the same level or at two consecutive levels is a convex MIMO.*

Proof. This follows by the definition of SMM: given a graph G, a MAXMISO $MM \subset G$ and a node $\overline{v} \in MM$ the SMMs of MM are the MAXMISOs of $G \setminus \overline{v}$. This means that if A and B are SMMs, then $A \cap B = \emptyset$. Therefore all the hypothesis of Theorem 1 are satisfied. ■

Basically the two approaches cluster optimally MAXMISOs at the same level [7], or heuristically at different levels [8], in convex MIMOs to implement in hardware reducing the execution time. Both approaches target the Molen organization [18] which allows for a virtually unlimited number of new instructions to be executed on the reconfigurable hardware, without limiting the number of input/output values.

Although the speed-up achieved by the two approaches is similar to the speed-up achieved for state-of-the-art algorithms for automatic instruction-set extension, the main limitation is represented by the MAXMISOs. The MAXMISOs are used as building block to generate convex MIMO instructions since they can be enumerated linearly with the number of nodes and they represent a trade off between quality of the solution and complexity of the approach. Nevertheless the speed-up is limited by a high number of inputs and outputs, on average, of the clusters selected for hardware implementation.

Every time a cluster undergoes a check to verify if a specific property is verified, the complexity of the approach increases. A limitation on the number of inputs and outputs of the clusters keeping a linear complexity, can then be obtained using SMMs instead of MMs. By Property 1 and 2 we know that SMMs can be enumerated with linear complexity in the number of nodes. *This means that the complexity of the two approaches does not increase if we use SMMs instead of MMs as building blocks to generate convex MIMO instructions.*

We can observe the following:

Remark 2. The partitioning of a graph in MAXMISOs generate a MMs-cover of the graph. Since every SUBMAXMISO is contained in a MAXMISO the

SMMs-cover is a refinement of the MMs-cover[4] [19]. This implies that the number of convex MIMO instructions which is possible to generate increases. More detailed, if there are n_1 MAXMISOs and n_2 SUBMAXMISOs with $n_2 = n_1 + \alpha$ and $\alpha > 0$, the additional MIMOs that is possible to generate are:

$$2^{n_1}(2^{\alpha} - 1). \tag{4}$$

5 Conclusions

In this paper, we have introduced an algorithm which enumerates with linear complexity in the number of processed elements, MISO instructions of variable size, and more specifically SUBMAXMISOs. These instructions can directly undergo a selection process for hardware-software partitioning or can be clustered with different policies for the generation of MIMO instructions. The algorithms can be included in an automatic design flow for the automatic generation of MIMO instructions as the ones proposed in [7,8]. In our future work we intend to verify with experimental results the benefit of the insertion of the SUBMAXMISOs generation algorithm in such a design flow. Moreover we aim to design and test additional algorithms for the generation of (convex) MIMO instructions.

References

1. Holmer, B.: Automatic Design of Computer Instruction Sets. PhD thesis, University of California, Berkeley (1993)
2. Huang, I., Despain, A.: Generating instruction sets and microarchitectures from applications. In: Proceedings of ICCAD '94 (1994)
3. Van Praet, J., Goossens, G., Lanneer, D., Man, H.D.: Instruction set definition and instruction selection for asips. In: Proceedings of ISSS '94 (1994)
4. Kastner, R., Kaplan, A., Memik, S.O., Bozorgzadeh, E.: Instruction generation for hybrid reconfigurable systems. ACM Trans. Des. Autom. Electron. Syst. 7(4), 605–627 (2002)
5. Atasu, K., Pozzi, L., Ienne, P.: Automatic application-specific instruction-set extensions under microarchitectural constraints. In: Proceedings of DAC '03 (2003)
6. Atasu, K., Dündar, G., Özturan, C.: An integer linear programming approach for identifying instruction-set extensions. In: Proceedings of CODES+ISSS '05 (2005)
7. Galuzzi, C., Panainte, E.M., Yankova, Y., Bertels, K., Vassiliadis, S.: Automatic selection of application-specific instruction-set extensions. In: Proceedings of CODES+ISSS '06 (2006)
8. Galuzzi, C., Bertels, K., Vassiliadis, S.: A linear complexity algorithm for the automatic generation of convex multiple input multiple output instructions. In: Proceedings of ARC 2007 (March 27-29, 2007)
9. Alippi, C., Fornaciari, W., Pozzi, L., Sami, M.: A dag-based design approach for reconfigurable vliw processors. In: Proceedings of DATE '99 (1999)

[4] A refinement of a cover C of X is a new cover D of X such that every set in D is contained in some set in C.

10. Cong, J., Fan, Y., Han, G., Zhang, Z.: Application-specific instruction generation for configurable processor architectures. In: Proceedings of FPGA '04 (2004)
11. Yu, P., Mitra, T.: Scalable custom instructions identification for instruction-set extensible processors. In: Proceedings of CASES '04 (2004)
12. Sun, F., Ravi, S., Raghunathan, A., Jha, N.K.: Synthesis of custom processors based on extensible platforms. In: Proceedings of ICCAD '02 (2002)
13. Brisk, P., Kaplan, A., Kastner, R., Sarrafzadeh, M.: Instruction generation and regularity extraction for reconfigurable processors. In: Proceedings of CASES '02 (2002)
14. Goodwin, D., Petkov, D.: Automatic generation of application specific processors. In: Proceedings of CASES '03 (2003)
15. Choi, H., Hwang, S.H., Kyung, C.M., Park, I.C.: Synthesis of application specific instructions for embedded dsp software. In: Proceedings of ICCAD '98 (1998)
16. Baleani, M., Gennari, F., Jiang, Y., Patel, Y., Brayton, R.K., Sangiovanni-Vincentelli, A.: Hw/sw partitioning and code generation of embedded control applications on a reconfigurable architecture platform. In: Proceedings of CODES '02 (2002)
17. Clark, N., Zhong, H., Mahlke, S.: Processor acceleration through automated instruction set customization. In: Proceedings of MICRO 36
18. Vassiliadis, S., Wong, S., Gaydadjiev, G., Bertels, K., Kuzmanov, G., Panainte, E.M.: The molen polymorphic processor. IEEE Trans. Comput. 53(11), 1363–1375 (2004)
19. Kosniowski, C.: A First Course in Algebraic Topology. Cambridge University Press, Cambridge (1980)