Two Algorithms for the Generation of Convex MIMO Instructions

Carlo Galuzzi, Koen Bertels, Stamatis Vassiliadis
Computer Engineering, EEMCS
Delft University of Technology
{carlo, k.l.m.bertels, s.vassiliadis}@ewi.tudelft.nl

Abstract— **In this paper, we address the Instruction-Set Extension problem. The problem boils down to identify clusters of operations that can be implemented as single complex operations in hardware which maximize some metric, typically performance. These new operations can then be incorporated in repertoire of the processor as new instructions.**

We propose a method to generate new convex instructions, where convexity is a property that guarantees a proper and feasible scheduling of the new instructions while respecting the dependencies. We propose two algorithms for the generation of convex Multiple Input Multiple Output (MIMO) instructions, given certain reconfigurable hardware resource constraints. The elementary building blocks of our approach are clusters of operations known as Maximal Multiple Input Single Output (MAXMISO) out of which the convex MIMO instructions are constructed.

The common starting point of both algorithms is the analysis of the application to identify the operations that can be clustered as MAXMISOs. These maximal MISOs are then combined to produce the set of new convex instructions. Whereas the first algorithm generates convex MIMO instructions formulating the problem as an ILP problem the second algorithm uses an heuristic to compose the new convex MIMO instructions. An important advantage of our approach is that it is not restricted to basic-block level nor does it impose any limitation on the number of the newly added instructions nor on the number of the inputs/outputs of these instructions.

Keywords: Instruction-Set Extension, Convexity, MIMO, Graph Algorithms.

I. Introduction

Instruction-Set Extension has been a major research topic in the last decade. The existing algorithms impose severe limitations on the number of input/output values as well as on the number of newly added instructions while their computational complexity can be exponential.

In this paper we introduce two algorithms that selects the new functionalities to be executed in hardware for improving the overall performance. The proposed algorithms targets the Molen organization [13] which allows for a virtually unlimited number of new instructions without limiting the number of input/output values of the function to be executed on the reconfigurable hardware. The elementary building blocks of the approach are clusters of operations known as Multiple Input Single Output (MISOs).

The common starting point of both algorithms is the analysis of the application to identify the largest MISOs, called MAXMISOs. These MAXMISOs are then combined and clustered as new application specific instructions. The result is a cluster of operations with Multiple Inputs and Multiple Outputs, called MIMO, which is executed on the reconfigurable hardware and which provides the maximum performance improvement under reconfigurable hardware resource constraints.

Whereas the first algorithm generates MIMO instructions formulating the problem as an ILP problem, the second algorithm uses an heuristic to compose the new MIMO instructions. The main contribution of this paper are:

- construct convex MIMO based on MAXMISOs clustering. Single MAXMISOs usually do not provide significant performance improvement. Thus, we propose MAXMISOs combinations in order to take advantage of the parallelism inherent to the hardware execution and the Theorem III.6 that guarantees the MIMO convexity by construction.
- elimination of the restrictions of the types and number of new instructions (in contrast with most of the existing approaches): there is no limitation on the number of input/output values or the number of new instructions.
- the proposed approaches are not restricted to basic-block level analysis but can be applied directly to large kernels.

The paper is organized as follows. In Section II, we discuss background information and related work. In the following section, we present the theoretical contribution. Finally in Section **??** we present conclusion and future work.

II. Background and related work

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 which are briefly presented in this section.

For the first category, a representative example is introduced in [1] 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 (four) processor cycles per newly added instruction. The approach presented in [9] 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. In consequence, heuristic and additional area constraints are introduced (e.g limitation of only 5-inputs is imposed) 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, it is removed generating a linear enumeration of them.

The algorithms included in the second category are more general and provide more significant performance improvement. However they also have exponential complexity. For example, in [4] the identification algorithm detects optimal convex MIMO subgraphs but the computational complexity is exponential. A similar approach described in [14] proposes the enumeration of all the instructions based on the number of inputs, outputs, area and convexity. The selection problem is not addressed. In [3] the authors target the identification of convex clusters of operations given input and output constraints. The clusters are identified with a ILP based methodology similar to the one we propose with the first algorithm. The main difference is that they iteratively solve ILP problems for each basic block, while in our approach we have one global ILP problem for the entire procedure. Additionally, the convexity is addressed differently: in [3], the convexity is verified at each iteration, while in our approach it is guaranteed by construction based on the Theorem III.6. Other approaches cluster operations considering the frequency of execution or the occurrence of specific nodes [11], [12] or regularity [6]. Still others impose limitation on the number of operands [5], [2], [8] and use heuristics to generate sets of custom instructions which therefore can not be globally optimal.

The algorithms we introduce in this paper combines concepts of both categories: first, MAXMISOs are identified as proposed in the first category, after which they are combined in convex MIMOs. This allows exploitation of the available parallelism provided by the hardware platform. Our algorithms as presented in this paper, require linear complexity for the MAXMISO enumeration. For the first algorithm, the MAXMISO combination is formulated as an integer linear problem whose optimal solution for most cases is found in a few seconds even for large (>5000 nodes) data flow graphs. For the second algorithm, the MAXMISO combination is performed with an heuristic of linear complexity. Additionally, the proposed algorithms do not impose any limitations on the number of input/output values (as in [2], [10], [7], [5]) or the number of newly added instructions.

III. Theoretical background

A. MISO properties and MAXMISO-clustering

Let $G = (V, E)$ be the dataflow graph of a given application, 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.

Let $\wp(G)$ be the power set of G and let n be the order of V[1]. The order of the power set of a set of order n is 2^n. Basically, there are two types of subgraphs that can be identified: graphs with Multiple Input Single Output (MISO) and graphs with Multiple Input Multiple Output (MIMO). Let G_{MISO} and G_{MIMO} be the subsets of $\wp(G)$ containing all MISOs and MIMOs respectively. The following chain of inclusions is valid:

$$G_{MISO} \subset G_{MIMO} \subset \wp(G). \quad (1)$$

In formal terms, a MISO is defined as follows.

[1] $\wp(G)$ is the set of all subgraphs of G, including the empty graph $\emptyset$ and G.

Definition III.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 $r \in V^*$ if $\forall\, v_i \in V^*$ there exists a path[2] $[v_i \rightarrow r]$, and every path $[v_i \rightarrow r]$ is entirely contained in G^*.*

By Definition III.1, A MISO is a connected graph. A MIMO, defined as the union of $m \geq 1$ MISOs can be either connected or disconnected.

Definition III.2. *A subgraph $G^* \subsetneq G$ is* **convex** *if there exists no path between two nodes of G^* which involves a node of $G \backslash G^*$*[3].

Convexity guarantees a proper and feasible scheduling of the new instructions which respects the dependencies. Definitions III.1 and III.2 imply that every MISO is a connected and convex graph. Even though these properties are not always satisfied by a MIMO, convex graphs have the following property:

Theorem III.3. *Let $G^* \subsetneq G$ be a convex subgraph of G. Then there exists $k \in \mathbb{N}$ such that*

$$G^* = \bigcup_{i=1}^{k} MISO_i. \tag{2}$$

Proof: Let G^* be a MISO. Then $k = 1$ and $MISO_1 = G^*$. Let G^* be a MIMO. Every node has single output[4] and therefore each node is trivially a MISO. This concludes the proof since every (sub)graph can be decomposed as the union of its nodes. ■

The previous theorem shows that an exhaustive enumeration of the MISOs contained in G gives the building blocks to generate all convex MIMOs. This faces with the theoretical exponential size of G_{MISO} and, by (1), of G_{MIMO}. A reduction in the number of building blocks influences the total number of convex MIMOs which it is possible to generate but it reduces the overall complexity of the generation process. This trade-off between complexity of the method and final goal can be achieved introducing a particular type of MISO, the MAXMISO. Let $\subset$ be the usual subset inclusion and $\wp(G)$ the power set of G. The couple $(\wp(G), \subset)$ is an ordered set and an element $G_{\overline{\imath}} \in \wp(G)$ is said to be **maximal** if, for all $G_i \in \wp(G)$, $G_{\overline{\imath}} \not\subset G_i$. A **maximal MISO** (MAXMISO) is a maximal element of $(G_{MISO}, \subset)$. A formal definition of MAXMISO is the following.

Definition III.4. *A MISO $G^*(V^*, E^*) \subset G(V, E)$ is a MAXMISO if $\forall v_i \in V \backslash V^*$, $G^+(V^* \cup \{v_i\}, E^+)$ is not a MISO.*

From the set-theory it is known that each element of G_{MISO} is either maximal (a MAXMISO) or there exists a maximal element containing it. [1] observed that if $A, B \in G_{MISO}$ are two MAXMISOs, then $A \cap B = \emptyset$. Since every node is trivially a MISO, the following equality holds:

$$G = \bigcup_{i \in I} MAXMISO_i,\ I \subset \mathbb{N}. \tag{3}$$

The empty intersection of two MAXMISOs implies that the MAXMISOs of a graph can be enumerated with linear complexity in the number of its nodes.

B. Convex MIMOs generation

Let $G_{MM} \subset G_{MISO}$ be the set of all MAXMISOs of G. Let $f : G_{MM} \subset G \rightarrow \hat{G}$ be the function that collapses the MAXMISOs of G in nodes of the graph $\hat{G}$, (Fig. III-A):

$$MM_i \in G_{MM} \subset G \mapsto a_i \in \hat{G}. \tag{4}$$

By the definition of f it follows that f is a surjective function. Two MAXMISOs cannot overlap and then f is injective and therefore bijective. Let f^{-1} be the inverse function of f: $f^{-1} : \hat{G} \rightarrow G_{MM}$, $a_i \in \hat{G} \mapsto MM_i \in G_{MM} \subset G$. f and f^{-1} are called the **collapsing** and **un-collapsing function** and $\hat{G}$ is called the the $MAXMISO$-**collapsed graph**.

Let $v \in V$ be a node of G and let $\textsc{Lev} : V \rightarrow \mathbb{N}$ be the function defined as follows:

- $\textsc{Lev}(v) = 0$, if v is an input node of G;
- $\textsc{Lev}(v) = \alpha > 0$, 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 III.5. *The level of $MAXMISO_i \in G$ is defined as follows:*

$$\textsc{Lev}(MAXMISO_i) = \textsc{Lev}(f(MAXMISO_i)). \tag{5}$$

Theorem III.6. *Let G be a DAG and $A_1, A_2 \subset G$ two MAXMISOs*[5]*. Let $\textsc{Lev}(A_1) \geq \textsc{Lev}(A_2)$ be the levels of A_1 and A_2 respectively. Let $C = A_1 \cup A_2$. If*

$$\textsc{Lev}(A_1) - \textsc{Lev}(A_2) \in \{0, 1\} \tag{6}$$

[2] A path is a sequence of nodes and edges, where the vertices are all distinct.

[3] G^* has to be a *proper subgraph* of G. A graph itself is always convex.

[4] G is the dataflow graph of a given application and the nodes represent assembler-like operations.

[5] Clearly $A_1 \cap A_2 = \emptyset$.

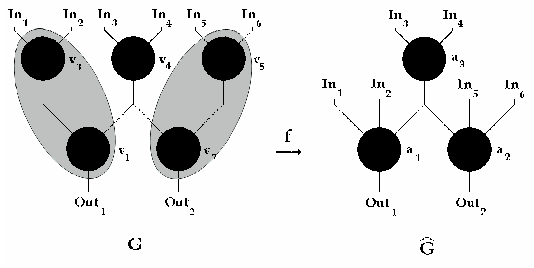

Fig. 1

The collapsing function $f : G \to \hat{G}$. $G_{MM} = \{MM_1, MM_2, MM_3\}$ where $MM_1 = \{v_1, v_3\}, MM_2 = \{v_4\}, MM_3 = \{v_2, v_5\}$. Then $MM_1 \mapsto a_1$, $MM_2 \mapsto a_3$ and $MM_3 \mapsto a_2$. In this case we have $G_{MISO} = \{v_1, v_2, v_3, v_4, v_5, MM_1, MM_3\}$ and clearly $G_{MM} \subsetneq G_{MISO}$.

then C is a convex MIMO. Moreover C is disconnected if the difference is 0.

Proof: Let G be decomposed as union of MAXMISOs and let f be the collapsing function. Let a_1, a_2 and c be the images through f of A_1, A_2 and C respectively. f transforms equation 6 in $\text{LEV}(a_1) - \text{LEV}(a_2) \in \{0, 1\}$. In both cases, by contradiction, if $c = a_1 \cup a_2$ is not convex, there exists at least one path from a_1 to a_2 involving a node a_k different from a_1 and a_2. Then

$$\text{LEV}(a_2) < \text{LEV}(a_k) < \text{LEV}(a_1). \tag{7}$$

Since by hypothesis $\text{LEV}(a_1) - \text{LEV}(a_2) \in \{0, 1\}$, it follows that $\text{LEV}(a_k) \notin \mathbb{N}$ which contradicts the assumption $\text{LEV}(\cdot) \in \mathbb{N}$. As a result c is a convex MIMO and then considering the un-collapsing function f^{-1}, $f^{-1}(c) = C = A_1 \cup A_2$ is a convex MIMO graph. In particular, if the difference is 0, c is disconnected. By contradiction if c is connected there exists a path from a_1 to a_2. Then $\text{LEV}(a_1) \neq \text{LEV}(a_2)$ which contradict the assumption $\text{LEV}(a_1) - \text{LEV}(a_2) = 0$. As a result c is disconnected and therefore $C = f^{-1}(c)$ is disconnected. ■

Corollary III.7. *Any combination of MAXMISOs at the same level or at consecutive levels is a convex MIMO.*

Proof: Let $C = A_1 \cup ... \cup A_k \subset G$ be the union of $k \geq 3$ MAXMISOs of G. Two scenarios are possible: $\text{LEV}(A_i) = \bar{\imath} \quad \forall i$, or $\text{LEV}(A_i) \in \{\bar{\imath}, \bar{\imath} + 1\}$. In both cases, let $c = f(C) = a_1 \cup ... \cup a_k$[6] be the image through f of the MAXMISOs-union. By contradiction if c is not convex, there exists al least a path between two nodes that is not included in c. This contradicts the previous Theorem III.6. Then c is a convex MIMO as well as $C = f^{-1}(c)$. ■

[6] $f(A \cup B) = f(A) \cup f(B)$.

C. The First Algorithm

The previous Theorem III.6 shows how to generate convex MIMO operations. This consists of two parts: the enumeration of all MAXMISOs in G and the combination of the MIMOs. Nevertheless, for the real newly added instructions, the total hardware resources are limited; in particular, we refer to the total available hardware area. A formal description of the clustering problem is as follows:

Problem statement. Given a graph $\hat{G} = \hat{G}(V, E)$ let d be the depth. Let HW and SW represent two disjoint sets of nodes such that $V = HW \cup SW$. Each node n is identified by two indices i, j where i is the level of the node and j is its position at level i. Let $l_{HW_{ij}}$ and $l_{SW_{ij}}$ be the latency of a node in HW and SW respectively. Let A and α_{ij} be the total available area and the area that a node n_{ij} occupies. Find the optimal subset $HW \subset V$ such that minimizes

$$\sum_{n_{ij} \in SW} l_{SW_{ij}} + \sum_{i=0}^{d} \max_{n_{ij} \in HW} l_{HW_{ij}}, \tag{8}$$

under the following constraint:

$$\sum_{n_{ij} \in HW} \alpha_{ij} \leq A. \tag{9}$$

Formula (8) represents the minimization of the total execution time: the first term is the execution time of the the MMs that are executed in software and have a sequential execution, while the second term represents the latency of the MMs that are selected for hardware execution in parallel at each level. The constraint expressed by (9) represents the requirement that all new instructions should fit on the total hardware area available.

The problem previously presented can be solved as

a $0-1$ linear programming problem to produce an optimal solution using an efficient solver.

0-1 Selection. Every node (MAXMISO) belongs to HW or SW sets. Consequently we associate to any $n_{ij} \in V$ a Boolean variable $x_{i,j}$ such that $x_{i,j} = 1$ if $n_{ij} \in HW$, 0 if $n_{ij} \in SW$, $i \in \{0, ..., d\}$ represents the level $Level(n_{ij})$, and j represents the position at level i. The search for the optimal subset HW is then the search of optimal 0/1 values for all x_{ij}.

Objective Function. Following the problem statement, formula (8) can be translated in the following objective function

$$\sum_{n_{ij} \in V} l_{SW_{ij}} * \overline{x_{ij}} + \sum_{i=0}^{d} \max_{n_{ij} \in V} l_{HW_{ij}} * x_{ij}. \quad (10)$$

If $x_{ij} = 1$ then $\overline{x_{ij}} = 0$ and consequently we can consider $n \in V$ given that $V = HW \cap SW$ and HW and SW are disjoint sets.

The *max* function included in the objective function transforms the problem into a non-linear problem, which is hard to be efficiently solved. In consequence, we transform the objective function by adding for each level a new integer variable $xmax$ which has the largest hardware latency of this level. More specifically, the objective function becomes:

$$\sum_{n_{ij} \in V} l_{SW_{ij}} * \overline{x_{ij}} + \sum_{i=0}^{d} xmax_i. \quad (11)$$

with the additional constraints:

$$xmax_i \geq l_{HW_{ij}} * x_{ij}, \ \forall i \in \{0, ..., d\} \text{with } n_{ij} \in Level_i. \quad (12)$$

Linear System of Inequalities. The original constraint given by (9) can be expressed as follow:

$$\sum_{n \in V} \alpha_{ij} * x_{ij} \leq A. \quad (13)$$

In summary, the steps required to generate the optimal set of convex MIMOs are the following:

- Step a: MAXMISO identification: using an algorithm similar to the one presented in [1]
- Step b: Construction of the reduced graph: each MAXMISO is collapsed on one node
- Step c: HW/SW estimation: evaluate the HW/SW execution latency for each MAXMISO
- Step d: ILP problem formulation: identify the objective function and set of constraints
- Step e: ILP problem solving: select the MAXMISOs which are combined into one new instruction

D. The Second Algorithm

Each convex MIMO is identified in two steps: first of all a cluster of nodes is grown within $\hat{G}$, the $MAXMISO$-collapsed graph, and afterward the cluster is extended with further nodes.

1^{st} **step**. Let $a_{\bar{\imath}}$ be a node of $\hat{G} = (\hat{V}, \hat{E})$ with $\text{LEV}(a_{\bar{\imath}}) = \alpha \in [0, d]$ and let $C = \{a_{\bar{\imath}}\}$. Let us define the following sets:

$$\begin{aligned}
\text{PRED}'(a_{\bar{\imath}}) &= \begin{cases} \{m \in \hat{V} \mid \text{LEV}(m) = \alpha - 1 \wedge \\ \quad \exists\, (m, a_{\bar{\imath}}) \in \hat{E}\} & \text{if } \alpha \geq 1 \\ \emptyset & \text{if } \alpha = 0 \end{cases} \\
\text{SUCC}'(a_{\bar{\imath}}) &= \begin{cases} \{m \in \hat{V} \mid \text{LEV}(m) = \alpha + 1 \wedge \\ \quad \exists\, (a_{\bar{\imath}}, m) \in \hat{E}\} & \text{if } \alpha \leq d - 1 \\ \emptyset & \text{if } \alpha = d \end{cases} \\
\text{SUCC}(a_{\bar{\imath}}) &= \begin{cases} \{m \in \hat{V} \mid \exists\, [a_{\bar{\imath}} \rightarrow m] \wedge \\ \quad \text{LEV}(m) > \text{LEV}(a_{\bar{\imath}})\} & \text{if } \alpha \leq d - 1 \\ \emptyset & \text{if } \alpha = d. \end{cases}
\end{aligned} \quad (14)$$

$C' = C \cup \text{PRED}'(a_{\bar{\imath}})$ is a convex MIMO. This holds for $\alpha \geq 1$ as a consequence of Theorem III.6 and for $\alpha = 0$ since a node is trivially a convex graph.

Let us consider $\text{SUCC}'(\text{PRED}'(a_{\bar{\imath}}))$ and let $N_{In}(n)$ and $N_{In_C}(n)$ be the number of inputs and the number of inputs coming from a set C of a node n. For each node n and each set C the following inequality can be satisfied:

$$2 * N_{In_C}(n) \geq N_{In}(n). \quad (15)$$

We define the following set:

$$C'' = \begin{cases} C' \cup \{n \in \text{SUCC}'(\text{PRED}'(a_{\bar{\imath}})) \mid (15) \text{ holds}\} & \text{if } n \text{ exists} \\ C' & \text{otherwise} \end{cases} \quad (16)$$

If there exists n such that (15) holds, by Theorem III.6, $C'' = C' \cup \{n\}$ is a convex MIMO. The property (15) is introduced to limit the total number of inputs of C''.

2^{nd} **step**. Let us define the following set:

$$\text{SUCC}(C'') = \bigcup_{m \in C''} \text{SUCC}(m) \quad (17)$$

For each $m \in \text{SUCC}(C'')$ such that

$$N_{In}(m) = N_{In_{C''}}(m), \quad (18)$$

$C'' \cup \{m\}$ is a convex MIMO. This follows from equation (18). In fact this implies that all inputs of m come from C'', avoiding the possibility to have a path between a node of C'' and m including a node not belonging to $C'' \cup \{m\}$. As a consequence $C''' =$

$C'' \cup \{m \in \text{SUCC}(C'') \mid N_{In_C}(m) = N_{In}(m)\}$ is a convex MIMO.

In summary, the steps required to generate the set of convex MIMOs are the following:

- Step a: MAXMISO identification: using an algorithm similar to the one presented in [1]
- Step b: Construction of the reduced graph: each MAXMISO is collapsed on one node
- Step c: HW/SW estimation: evaluate the HW/SW execution latency for each MAXMISO
- Step d: Until the area constraint is violated, choose a node of $\hat{G}$, grow C''' and remove the nodes of C''' from the node to further analyze.

Remark III.8. *By Step d, it follows that the convex MIMOs are generated linearly with the number of node of $\hat{G}$.*

IV. Conclusions

In this paper, we have introduced two algorithms which select clusters of MAXMISO for execution as a new application-specific instruction on the reconfigurable hardware. The first algorithm formulates the instruction selection as an ILP problem for the minimization of the execution time under hardware area constraints. The second algorithm combines instructions with a linear complexity heuristic based on Theorem III.6. In our future work we intend to introduce more general MAXMISO clustering and more general operation clustering.

References

[1] C. Alippi, W. Fornaciari, L. Pozzi, and M. Sami. A DAG-Based Design Approach for Reconfigurable VLIW Processors. In *Proceedings of DATE 1999*, pages 778–779, Munich, Germany, March 1999.

[2] M. Arnold and H. Corporaal. Design Domain Specific Processors. In *Proceedings of the 9th International Workshop on Hardware/Software CoDesign*, pages 61–66, April 2001.

[3] K. Atasu, G. Dündar, and C. Özturan. An Integer Linear Programming Approach for Identifying Instruction-Set Extensions. In *Proceedings of CODES+ISSS'05*, pages 172–177, New Jersey, USA, September 2005.

[4] K. Atasu, L. Pozzi, and P. Ienne. Automatic Application-Specific Instruction-Set Extensions under Microarchitectural Constraints. In *Proceedings of 40th DAC*, pages 256–261, Anaheim, California, June 2003.

[5] M. Baleani, F. Gennari, Y. Jiang, Y. Pate, R. K. Brayton, and A. Sangiovanni-Vincentelli. HW/SW Partitioning and Code Generation of Embedded Control Application on a Reconfigurable Architecture Platform. In *Proceedings of the 10th International Workshop on Hardware/Software Codesign*, pages 151–156, Estes Park, Colo., May 2002.

[6] P. Brisk, A. Kaplan, R. Kastner, and M. Sarrafzadeh. Instruction Generation and Regularity Extraction for Reconfigurable Processors. In *Proceedings of CASES 2002*, pages 262 – 269, Grenoble, France, 2002.

[7] H. Choi, J. S. Kim, C. W. Yoon, I. C. Park, S. H. Hwang, and C. M. Kyung. Synthesis of Application Specific Instructions for Embedded DSP Software. *IEEE Transactions on Computers*, 48(6):603–614, June 1999.

[8] N. Clark, H. Zhong, and S. Mahlke. Processor Acceleration Through Automated Instruction Set Customization. In *Proceedings of the 36th MICRO*, pages 129–140, December 2003.

[9] J. Cong, Y. Fan, G. Han, and Z. Zhang. Application Specific Instruction Generation for Configurable Processor Architectures. In *Proceedings of FPGA'04*, pages 183–189, Monterey, California, February 2004.

[10] D. Goodwin and D. Petkov. Automatic Generation of Application Specific Processors. In *Proceedings of CASES'03*, pages 137–147, San Jose, California, 30 Oct. - 1 Nov. 2003.

[11] R. Kastner, A. Kaplan, S. Ogrenci Memik, and E. Bozorgzadeh. Instruction Generation for Hybrid Reconfigurable System. *ACM Transactions on Design Automation of Embedded Systems*, 7(4):605–627, October 2002.

[12] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha. Synthesis of Custom Processors Based on Extensible Platforms. *Proceedings of ICCAD 2002*, pages 641–648, November 2002.

[13] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G.K. Kuzmanov, and E. Moscu Panainte. The Molen Polymorphic Processor. *IEEE Transactions on Computers*, 53(11):1363– 1375, November 2004.

[14] P. Yu and T. Mitra. Scalable Custom Instructions Identification for Instruction-Set Extensible Processors. In *Proceedings of CASES'04*, pages 69–78, 2004.