# The Spiral Search: A Linear Complexity Algorithm for the Generation of Convex MIMO Instruction-Set Extensions

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 decade and it consists of 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. A tradeoff between complexity and quality of the solution can be achieved limiting this search to implementable instructions. In this paper we propose a linear complexity algorithm for the generation of convex Multiple Input Multiple Output (MIMO) instructions of variable size based on the notion of spiral. Convex implementable MIMO clusters of instructions are identified by means of a spiral search through the levels of a graph. These new instructions can be directly selected or combined for more complex Instruction-Set extensions. An important feature of our algorithm is that it is neither restricted to basic-block level nor it imposes any limitation on the number of the newly instructions nor on the number of the inputs/outputs of these instructions.**

## I. Introduction

Along the last decade, we have witnessed an increasing popularity of reconfigurable architectures thanks to the capability to provide high overall performances of execution of an application, by tuning the architecture towards the specific requirements of the application. More and more often this is achieved via the automatic extension of a given Instruction-Set (IS) with new customized instructions for the specific application.

An example of reconfigurable architecture can be realized by combining a GPP and a reconfigurable hardware as an FPGA (see, for example, [?]). When an application is executed on a traditional architecture, a certain number of instructions, the ones belonging to the Instruction-Set, are executed in hardware while the rest of the instructions are executed in software. A software execution is typically more costly in terms of execution time than the ones executed in hardware. When the same application is executed on a reconfigurable architecture, a remarkable speedup in the execution of the application can be obtained identifying within the application clusters of instructions with peculiar properties which when implemented as a single complex instruction on the reconfigurable hardware reduce the total execution time. Once these new complex instructions are hardwired, they represent an extension of the given Instruction-Set. Additionally, the reconfigurability of the architecture allows the creation of new complex instructions ad hoc, that is Application-Specific Instruction-Set Extensions.

Enumerating all possible clusters of instructions within an application exhaustively is not computationally feasible due to the exponential number of candidates. Although human ingenuity in manual creation of custom capabilities creates high quality results, performance and time-to-market requirements as well as the growing complexity of the design space, can benefit from an automatic design flow for the generation and use of these new capabilities. Moreover the selection of multiple custom instructions from a large set of candidates involves complex tradeoff and can be difficult to be performed manually.

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

In this paper we propose a linear complexity algorithm based on the notion of spiral for the generation of MIMO instructions. Implementable MIMO clusters of instructions are identified by means of a spiral search through the level of a graph. These clusters can directly undergo a selection process for hardware-software partitioning or can be clustered with different policies for the generation of more complex MIMO instructions. Although Instruction-Set extension is the combination of instruction generation and instruction selection, in this paper only instruction generation is addressed. Different selection criteria can be used depending on the target architecture. 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 algorithm generates implementable MIMO instructions of variable size suitable for inclusion in a design flow for automatic selection of MIMO instructions. Our approach requires linear complexity in the number of processed elements as proven in Section **??**.
- elimination of the restrictions of the type and number of new instructions (in contract with most of the existing approaches): there is no limitation on the number of input/output values or on the number of new instructions.
- 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 II, background information and related works are provided. In Sections III and **??**, the basic definitions and the algorithm for MIMO instruction generation are presented respectively. Concluding remarks and an outline of research conducted are given in Section **??**.

## II. 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 [?] 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 some processor cycles per newly added instruction. The approach presented in [?] 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 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 [**?**] the identification algorithm detects optimal convex MIMO subgraphs but the computational complexity is exponential. A similar approach described in [**?**] proposes the enumeration of all the instructions based on the number of inputs, outputs, area and convexity. The selection problem is not addressed. In [**?**] 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 [**?**]. The main difference is that in [**?**] the authors iteratively solve ILP problems for each basic block, while in [**?**] the authors have one global ILP problem for the entire procedure. Additionally, the convexity is addressed differently: in [**?**], the convexity is verified at each iteration, while in [**?**] it is guaranteed by construction. Other approaches cluster operations by considering the frequency of execution or the occurrence of specific nodes [**?**], [**?**] or regularity [**?**]. Still others impose limitation on the number of operands [**?**], [**?**], [**?**], [**?**] 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 spiral. Starting from a directed acyclic graph representing an application, the nodes are divided per levels and connected convex MIMO clusters of instructions are identified by means of a spiral search through the levels of the graph. Theorem III.3 and Section **??** guarantee the MIMO convexity by constructions. Moreover the algorithm has linear complexity in the number of processed elements, as shown by Remark **??**. This algorithm together with the one presented in [**?**] define two methods for the generation of MIMO and MISO instructions of variable size.

## III. Theoretical background

### A. 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 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 $r \in V^*$ provided that $\forall\ v_i \in V^*$ there exists a path*[1] *$[v_i \to r]$, and every path $[v_i \to r]$ 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. 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 $G_{MIMO}$.

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

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 implementable MIMO graphs, i.e. MIMO graphs with specific properties which make the graph suitable for an hardware implementation. One of these properties is the **convexity**, as outlined in the next Section III-B.

### B. Convex MIMO graphs

**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. MIMOs can be convex or not [**?**], [**?**], [**?**].

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

- $\text{Lev}(v) = 0$, if $v$ is an input node of $G$;
- $\text{Lev}(v) = \alpha > 0$, if there are $\alpha$ nodes on the longest path from $v$ and the level 0 of the input nodes.

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

In [**?**], [**?**], the authors have presented two theorems for the generation of convex MIMO graphs based on maximal MISO (MAXMISO) combinations. These theorems can be resumed as follows.

**Theorem III.3.** *Let $G(V, E)$ be a DAG and let $v_1, v_2 \in V$ be two nodes of $G$. Let $\text{Lev}(v_1) \geq \text{Lev}(v_2)$ be the levels of $v_1$ and $v_2$ respectively. Let $A = v_1 \cup v_2$. If*

$$\text{Lev}(v_1) - \text{Lev}(v_2) \in \{0, 1\} \tag{1}$$

*then $A$ is a convex MIMO. Moreover*
*$(a)$ $A$ is disconnected if the difference is 0.*
*$(b)$ Any combination of nodes at the same level or at two consecutive levels is a convex MIMO.*

Section III-C and **??** show in detail how the algorithm generates convex MIMO graphs with linear complexity in the number of processed elements.

### C. The spiral search

The algorithm we propose is based on the notion of spiral. We now briefly describe what is a spiral. The easiest spiral line is the one studied by Archimedes, which has his name: the *Archimedean Spiral*[4]. This spiral is generated when a point $P$ moves with constant speed $v$ on a line which, in turn, rotates around one of his points $O$ with constant angular velocity $\omega$. The point $O$ is called **center of the spiral**. The distance $T_d$ between two consecutive turns is called **turn distance**.

Let $\text{Lev}_1, \ldots \text{Lev}_d$ be the levels of the nodes of a graph $G$ and let $O$ be a node of $G$ with $\text{Lev}(O) = i$.

**Definition III.4.** *If $O$ is a seed node*[5], *a **spiral search** is a clustering search which looks for nodes to cluster starting from the level of the*

[2]$G_{MISO} = \{G^* \subset G,\ s.t.\ N_{Out} = 1\} \subset \{G^* \subset G,\ s.t.\ N_{Out} \geq 1\} = G_{MIMO}$.

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

[4]Achimedes, *"On Spirals"*, 225 BC.

[5]A seed node is a node selected as initial node in the generation of a cluster.

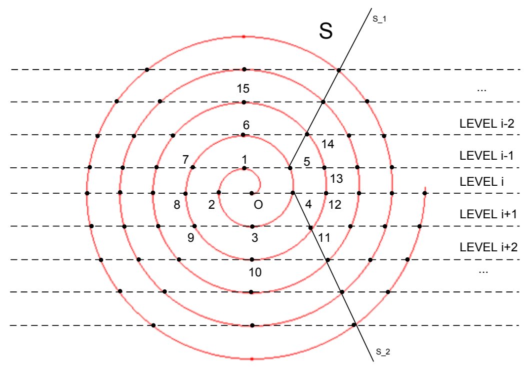

Figure 1. The spiral search. If $O \in L_i$ is a seed node, the levels are analyzed following the order of intersection between the spiral and the levels: $1, 2, 3, \ldots$.

*seed and following a spiral path $S$ with center the seed, every time the spiral intersects a level, the nodes of that level (some or all) are analyzed and the ones which respect a predefined property*[6] *$P$ are included in the cluster.*

Figure **??** depicts a spiral search, with turn distance equals to one level, in which the levels are analyzed following the order of intersection between the spiral and the levels.

## IV. The Spiral Search Algorithm

In order to formally describe our algorithm, we first define some sets. Let $a_{\overline{\iota}}$ be a node of $G = (V, E)$ with $\text{Lev}(a_{\overline{\iota}}) = \alpha \in [0, d]$. It is possible to define the following sets:

$$
\begin{aligned}
\text{pred}'(a_{\overline{\iota}}) &= \begin{cases} \{m \in V \mid \text{Lev}(m) = \alpha - 1 \wedge \\ \quad \exists\, (m, a_{\overline{\iota}}) \in E\} & \text{if } \alpha \geq 1 \\ \emptyset & \text{if } \alpha = 0 \end{cases} \\
\text{succ}'(a_{\overline{\iota}}) &= \begin{cases} \{m \in V \mid \text{Lev}(m) = \alpha + 1 \wedge & \text{if } \alpha \leq d - 1 \\ \quad \exists\, (a_{\overline{\iota}}, m) \in E\} \\ \emptyset & \text{if } \alpha = d \end{cases}
\end{aligned} \tag{2}
$$

Let $A$ be a cluster of nodes of $G$ with $h \leq \text{Lev}(n) \leq k$ for every node $n \in A$. We have the following:

$$
\begin{aligned}
\text{succ}^*(A) &= \begin{cases} \{m \in V \backslash V_A \mid \exists\, [n \rightarrow m] \wedge & \text{if } k \leq d - 1 \\ \quad h < \text{Lev}(m) \leq k + 1\} \\ \{m \in V \backslash V_A \mid \exists\, [n \rightarrow m] \wedge & \text{if } k = d \\ \quad h < \text{Lev}(m) \leq k\} \end{cases} \\
\text{pred}'(A) &= \begin{cases} \{m \in V \backslash V_A \mid \text{Lev}(m) = h - 1 \wedge & \text{if } h \geq 1 \\ \quad \exists\, (m, n) \in E\} \\ \{m \in V \backslash V_A \mid \text{Lev}(m) = h \wedge & \text{if } h = 0 \\ \quad \exists\, (m, n) \in E\} \end{cases}
\end{aligned} \tag{3}
$$

The algorithm proposed in this paper generate each convex MIMO through multiple steps by means of a spiral search through the levels of the graph. We now describe in detail all the steps necessary for the generation of the cluster as well as the theoretical proof of the cluster convexity at each step. We set $T_d$ equals to 1 level.

**STEP 1**. Let $a_{\overline{\iota}}$ be a node of $G = (V, E)$ chosen as a seed node (center of the spiral) with $\text{Lev}(a_{\overline{\iota}}) = i \in [0, d]$. Let $C = \{a_{\overline{\iota}}\}$. $C' = C \cup \text{pred}'(a_{\overline{\iota}})$ is a convex MIMO. This holds for $\alpha \geq 1$ as a consequence of Theorem III.3 and for $\alpha = 0$ since a node is trivially a convex graph. This step corresponds to intersection 1 in Figure **??**.

**STEP 2**. Let us consider $\text{succ}'(\text{pred}'(a_{\overline{\iota}}))$ and let $N_{In}(n)$ be the number of inputs of a node $n$ and let $N_{In_C}(n)$ be 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[7]:

$$
2 * N_{In_C}(n) \geq N_{In}(n). \tag{4}
$$

This can be reformulated by saying that the number of inputs of $n$ coming from the set $C$ has to be greater than or equal to at least half the total inputs of $n$. We define the following set:

$$
C'' = \begin{cases} C' \cup \{n \in \text{succ}'(\text{pred}'(a_{\overline{\iota}})) \mid (\textbf{??}) \text{ holds}\} & \text{if } n \text{ exists} \\ C' & \text{otherwise.} \end{cases} \tag{5}
$$

If there exists $n$ such that (**??**) holds, by Theorem III.3, $C'' = C' \cup \{n\}$ is a convex MIMO. This step corresponds to intersection 2 in Figure **??**.

**STEP 3**. For each $m \in \text{succ}^*(C'')$ such that

$$
N_{In}(m) = N_{In_{C''}}(m), \tag{6}
$$

$C'' \cup \{m\}$ is a convex MIMO. This follows from (**??**). If the total number of inputs of $m$ is equal to the number of inputs coming from $C''$, it follows that it does not subsist the possibility of having a path between a node of $C''$ and $m$ which includes a node not belonging to $C'' \cup \{m\}$. As a consequence

$$
C''' = C'' \cup \{m \in \text{succ}^*(C'') \mid N_{In_C}(m) = N_{In}(m)\} \tag{7}
$$

is a convex MIMO. This step corresponds to intersection 3 in Figure **??**.

The algorithm analyzes the nodes of the graph following a spiral search through the levels of the graphs. As described in Section III-C and depicted in Figure **??**, the levels are analyzed following the order of intersection which depends on the turn distance. Since $T_d = 1$, the next intersections to analyze would be 4 and then 5, i.e. level $i$ and level $i - 1$, to look for nodes such that Equation **??** holds. We have the following:

**Theorem IV.1.** *Intersections 4 and 5 do not provide any node $p \notin C'''$ such that $C''' \cup \{p\}$ is convex and $N_{In}(p) = N_{In_{C'''}}(m)$.*

*Proof:* Let us consider intersection 4. By contradiction let $p$ be a node such that $C''' \cup \{p\}$ is convex with $p$ belonging to level $i$. Two scenarios are possible: there exists at least a path between $p$ and one node at level $i + 1$ or between one node at level $i - 1$ and $p$. The first case is not possible as the nodes at level $i + 1$ belongs to $C'''$ if and only if they have all inputs coming from $C''$. If $p$ exists, it means that there exists at least one node in $C'''$ at level $i + 1$ with one input not coming from $C''$. In the second case all nodes $p$ such that $C''' \cup \{p\}$ is convex and $N_{In}(p) = N_{In_{C'''}}(p)$ have been already included during STEP 2. Therefore no additional node can be further included in the cluster.

Let us consider intersection 5. Clearly no node at level $i - 1$ can have all inputs such that $N_{In}(p) = N_{In_{C'''}}(p)$ since the inputs of $p$ come from levels lower than or equal to $i - 2$. ■

Following similar arguments, it is possible to show, see Figure **??**, that all the intersections in the region bounded by the two half-lines $s_1$ and $s_2$ do not contain nodes suitable for an inclusion in the convex cluster.

**STEP 4**. By Theorem **??** and by the properties of the spiral search, the next level to analyze is level $i - 2$. In general we have the following:

**Remark IV.2.** *If the level analyzed in STEP 3 is $i + \beta$ and the center of the spiral is on level $i$, the next level to analyze is the level symmetric respect to the center of the spiral decreased by one:*

$$
\text{Level} = i - [(i + \beta) - i] - 1 = i - \beta - 1. \tag{8}
$$

[6]This property can be local, i.e. related to the single level, as well as it can be global, i.e. related to all levels.

[7]This inequality holds also for different type of analysis, where the granularity of the analysis can be fine-grained as well as coarse-grained (see Remark **??**).

*If this level does not exist, we consider the lower analyzed so far.*

The algorithm repeats STEP 1-3 till the final connected convex MIMO cluster is generated. Additionally, in STEP 1 we now consider $C = C'''$ and, as a consequence, we do not consider $C' = C \cup \text{PRED}'(a_{\bar{\iota}})$ but $C' = C \cup \text{PRED}'(C''')$.

In Figure **??**, we present an example that shows the way by which the algorithm generates the convex MIMO cluster.

**Remark IV.3.** *The algorithm generates at each step a convex MIMO cluster. One of the main qualities of this algorithm is its adaptability. As it has been shown in the previous steps the convexity of the cluster is independent by the choice of the seed. This means that depending on the application and/or on the target, the seed node can be selected as a node with specific properties like area or power consumption below a certain threshold previously defined or with a limited number of inputs/outputs, etc.*

**Remark IV.4.** *The limitation on the number of inputs/outputs of the nodes of the graph to analyze introduced in Section III-A can be removed depending on the level of abstraction on which the algorithm has to operate. As it has been shown during the previous steps, restrictions on the number of Inputs/Outputs never represent a point of discussion. This means that the algorithm works properly with a fine-grained as well as a coarse-grained level of analysis, where nodes can have a different number of inputs and/or outputs, provided that the graph to analyze is a directed acyclic graph.*

**Remark IV.5.** *The algorithm can be adapted to different needs. Future work will address the description of the parameters that can be introduced to adapt the algorithm to specific limitations like limited number of inputs/outputs, maximum delay and limited hardware resources like the area. Additionally, every time a cluster is generated, its nodes are removed from the node to further analyze. This implies that the algorithm has linear complexity in the number of the nodes analyzed.*

## V. Conclusions

In this paper, we have presented an algorithm for the generation of convex and connected Multiple Input Multiple Output (MIMO) instructions. Implementable MIMO clusters of instructions are identified by means of a spiral search through the level of a graph. These clusters can directly undergo a selection process for hardware/software partitioning or can be clustered with different policies for the generation of more complex MIMO instructions. The algorithm has linear complexity with the number of processed elements and the cluster convexity is guaranteed by construction, as shown in the paper. Additionally, the algorithm can include a certain number of parameters, for the limitation of the size and shape of the final cluster. In our future work, we intend to provide experimental results to show the qualities of the algorithm, here just proved theoretically, and we aim to design and test additional algorithms for the generation of (convex) MIMO instructions.

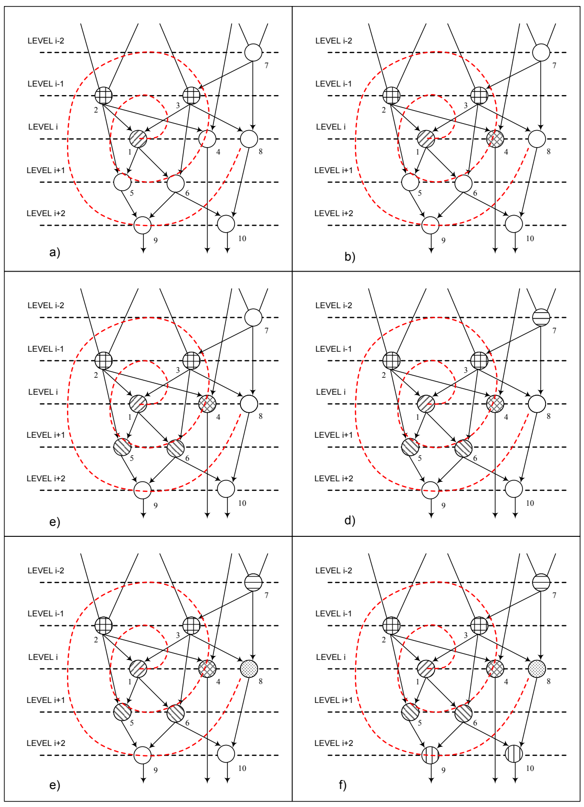

Figure 2. Example of an application of the algorithm. $C = \{1\}$, **a)** $C' = \{1, 2, 3\}$, **b)** $C'' = \{1, ..., 4\}$, **c)** $C''' = \{1, ..., 6\}$. After **d)** the algorithm restarts STEP 1-3 as described in Section **??**, we have then **d)**, **e)** and **f)** which generate the cluster $\{1 - 10\}$. The algorithm, then, continues the expansion of the cluster if there are nodes to be considered for extension.

## References

[1] C. Alippi, W. Fornaciari, L. Pozzi, and M. Sami. A dag-based design approach for reconfigurable vliw processors. In *Proc. of DATE '99*.

[2] K. Atasu, G. Dündar, and C. Özturan. An integer linear programming approach for identifying instruction-set extensions. In *Proc. of CODES+ISSS '05*.

[3] K. Atasu, L. Pozzi, and P. Ienne. Automatic application-specific instruction-set extensions under microarchitectural constraints. In *Proc. of DAC '03*.

[4] M. Baleani, F. Gennari, Y. Jiang, Y. Patel, R.K. Brayton, and A. Sangiovanni-Vincentelli. Hw/sw partitioning and code generation of embedded control applications on a reconfigurable architecture platform. In *Proc. of CODES '02*.

[5] P. Brisk, A. Kaplan, R. Kastner, and M. Sarrafzadeh. Instruction generation and regularity extraction for reconfigurable processors. In *Proc. of CASES '02*.

[6] H. Choi, S.H. Hwang, C.-M. Kyung, and I.-C. Park. Synthesis of application specific instructions for embedded dsp software. In *Proc. of ICCAD '98*.

[7] N. Clark, H. Zhong, and S. Mahlke. Processor acceleration through automated instruction set customization. In *Proc. of MICRO 36*.

[8] J. Cong, Y. Fan, G. Han, and Z. Zhang. Application-specific instruction generation for configurable processor architectures. In *Proc. of FPGA '04*.

[9] C. Galuzzi, K. Bertels, and S. Vassiliadis. A linear complexity algorithm for the generation of multiple inputs single output instructions of variable size. In *Proce. of SAMOS VII Workshop*, pages 283–293, July 2007.

[10] C. Galuzzi, K. Bertels, and S. Vassiliadis. A linear complexity algorithm for the automatic generation of convex multiple input multiple output instructions. In *Proc. of ARC 2007*, 27-29 March 2007.

[11] C. Galuzzi, E. Moscu Panainte, Y. Yankova, K. Bertels, and S. Vassiliadis. Automatic selection of application-specific instruction-set extensions. In *Proc. of CODES+ISSS '06*.

[12] D. Goodwin and D. Petkov. Automatic generation of application specific processors. In *Proc. of CASES '03*.

[13] P. Ienne and R. Leupers. *Customizable Embedded Processors: Design Technologies and Applications (Systems on Silicon)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[14] R. Kastner, A. Kaplan, S. Ogrenci Memik, and E. Bozorgzadeh. Instruction generation for hybrid reconfigurable systems. *ACM Trans. Des. Autom. Electron. Syst.*, 7(4):605–627, 2002.

[15] F. Sun, S. Ravi, A. Raghunathan, and N.K. Jha. Synthesis of custom processors based on extensible platforms. In *Proc. of ICCAD '02*.

[16] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. Moscu Panainte. The molen polymorphic processor. *IEEE Trans. Comput.*, 53(11):1363–1375, 2004.

[17] P. Yu and T. Mitra. Scalable custom instructions identification for instruction-set extensible processors. In *Proc. of CASES '04*.