Algorithms for the Automatic Extension of an Instruction-Set

Carlo Galuzzi, Dimitris Theodoropoulos, Roel Meeuws and Koen Bertels
Computer Engineering, Delft University of Technology, The Netherlands
{C.Galuzzi, D.Theodoropoulos, K.Bertels, R.Meeuws}@tudelft.nl

***Abstract*—In this paper, two general algorithms for the automatic generation of instruction-set extensions are presented. The basic instruction set of a reconfigurable architecture is specialized with new application-specific instructions. The paper proposes two methods for the generation of convex multiple input multiple output instructions, under hardware resource constraints, based on a two-step clustering process. Initially, the application is partitioned in single-output instructions of variable size and then, selected clusters are combined in convex multiple output clusters following different policies. Our results on well-known kernels show that the extended Instructions-Set allows to execute applications more efficiently and needing fewer cycles. Our results show that a significant overall application speed-up is achieved even for large kernels (for ADPCM decoder the speed-up is up to x2.2 and for TWOFISH encoder the speedup is up to x5.5).**

I. Introduction

In the past decade, we have witnessed a general shifting from the use of general-purpose computing architecture to architecture able to perform only a limited number of tasks but more efficiently. Although general-purpose architecture can execute a broad range of applications making them extremely flexible, the power consumption is relatively high. A good trade-off between flexibility and power consumption is introduced by reconfigurable systems. A simple reconfigurable architecture can be realized, for instance, by coupling a General Purpose Processor (GPP) and a reconfigurable hardware like an FPGA. When an application is executed on a general purpose architecture, a certain number of instructions are executed in hardware, namely the ones that belong to the Instruction-Set, whereas the rest of the instructions are executed in software. If the same application is executed on a reconfigurable architecture, we can use the reconfigurable hardware to implement and execute additional, more complex, *application-specific instructions*, so as to extend the Instruction-Set and speed-up the execution of the application on the system. The identification of those instructions suitable for hardware implementation represents the so-called *Instruction-Set Extension (ISE) problem*.

Taking into account the data-flow or control-flow graph of an application, it is easy to understand that the parts of the application suitable for hardware implementations correspond to subgraphs of the graph representing the application. The subgraph enumeration problem is a well-known problem which is computationally complex and requires exponential time to provide an exhaustive enumeration of all the subgraphs. Since not all subgraphs are suitable for a hardware implementation, the problem becomes the design of efficient algorithms for the identification of those instructions suitable for a hardware implementation.

In this context, we present two general algorithms for the automatic identification of convex[1] Multiple Input Multiple Output (MIMO) instruction-set extensions. The new instructions are generated via a two-step process which, initially, partitions the application in single-output subgraphs of variable size. After that, selected subgraphs are combined in convex multiple output clusters following different policies. Our results show that the extended Instruction-set allows to execute applications more efficiently and needing fewer cycles (see Section V). More specifically, the main contributions of this paper are the following:

- the construction of convex MIMO instructions through the combination of single-output clusters of instructions of variable size. In contrast with existing approaches, the convexity of the final cluster is guaranteed by construction and does not require additional checks of the clusters reducing, in this way, the overall computational complexity.
- an overall linear complexity of the proposed solutions. The generation of instruction-set extensions is comparable to the subgraph enumeration problem that is a well known computationally complex problem. Our algorithms heuristically generate instruction-set extensions in two steps of linear complexity.
- the proposed approach can be directly applied to large kernels as suggested by the experiments carried out on the MOLEN polymorphic processor prototype [21], [1].

The paper unfolds as follows. In Section II, background information and related works are provided. In Section III and IV, the context is further formalized and the theoretical contribution is presented. Section V presents the experimental results. Concluding remarks and an outline of the research conducted are given in Section VI.

II. Background and Related Work

There are two types of clusters that can be identified within a graph, based on the number of output values: Multiple Input Single Output (MISO) and Multiple Input Multiple Output (MIMO). Accordingly there are two types of algorithms for achieving Instruction-Set extension.

Concerning the first category, representative examples are presented in [2], [8], [9]. The exponential number of MISOs within a graph turns into an exponential complexity of the algorithm used for their enumeration. [2] reduces the complexity of the analysis generating only MISO instructions of maximal size, called MAXMISO (MM). The algorithm exhaustively enumerates all MMs with linear complexity in the number of processed elements. [8] reduces the complexity with the use of a heuristic and additional area constraints. A different approach is presented in [9] where, with an iterative application of the MM clustering presented in [2], MISO instructions called SUBMAXMISOs (SMMs) are generated with linear complexity in the number of processed elements.

The algorithms included in the second category are more general and provide more significant performance improvement. However they also have exponential complexity in the general case. The approaches presented in [13], [4], [6] generate and select optimal convex MIMO instructions but the computational complexity is exponential. In [24], [25], the authors address the enumeration of all connected and disconnected patterns based on the number of inputs, outputs, area and convexity. In [18], the authors present a complete, generic processor customization flow which uses a novel identification algorithm. It generates custom instructions through a

[1]The convexity of a graph, property described in Section III, guarantees a proper and feasible scheduling of the instruction generated which respects the dependencies.

978-3-9810801-5-5/DATE09 © 2009 EDAA

technique which makes use of local registers to overcame the I/O constraints on the new custom instructions.

In [3] and [13] the authors address the generation of convex clusters of operations with an ILP-based methodology. Other works impose limitations on the number of operands [5], [7] and use heuristics to generate sets of custom instructions which therefore can not be globally optimal. Still, other works [15], [20], [22] cluster operations considering the frequency of execution or the occurrence of specific nodes.

In this paper, we combine concepts of both categories: initially, the application is partitioned in single-output clusters of instructions of variable size. After that, selected clusters are combined for the generation of convex MIMO instructions, following different policies in clustering.

III. Theoretical Background

We assume that the input dataflow graph is a Directed Acyclic Graph (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 in a graph: MISOs and MIMOs.

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[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. MIMOs can be convex or not. An exhaustive enumeration of the MISOs contained in G gives all the necessary building blocks to generate all possible convex MIMOs. This deals with the exponential number of MISOs, and therefore MIMOs, contained in G whose enumeration requires a solution of exponential complexity in the number of processed elements. A reduction of the number of the building blocks reduces the total number of convex MIMOs which it is possible to generate. Anyhow, it 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 of maximal size, the MAXMISOs (MMs).

Definition III.3 A MISO $G^*(V^*, E^*) \subset G(V, E)$ is a MM if $\forall v_i \in V \backslash V^*$, $G^+(V^* \cup \{v_i\}, E^+)$ is not a MISO.

[2] observed two properties related to MMs: first, every MISO is either a MM or there exists a MM containing it. Second, if A, B are two MMs then $A \cap B = \emptyset$. The empty intersection of MMs implies that the MMs of a graph can be enumerated with linear complexity in the number of its nodes and the set of all MMs represents a minimal cover.

[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.

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;
- $\textsc{Lev}(v) = \alpha > 0$, 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.4 The level of a MM $MM_i \in G$ is defined as follows:

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

where $f : G \rightarrow \hat{G}$ is the collapsing function, the function which collapses the MMs of G in nodes of the graph $\hat{G}$ [4].

Let us consider a MM MM_i. Each node $v_j \in MM_i$ belongs to level $\textsc{Lev}(v_j)$. Let $\overline{v} \in MM_i$, with $0 \leq \textsc{Lev}(\overline{v}) \leq d$. If we apply the MM algorithm to $MM_i \setminus \{\overline{v}\}$, each MM identified in the graph is called a SUBMAXMISO (SMM) of $MM_i \setminus \{\overline{v}\}$ (or, shortly, of MM_i). The set of the SMMs tightly depends on the choice of $\overline{v}$. For example $\overline{v}$ can be either an exit node, or an inner node randomly chosen, 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 MM. The target of this paper is the generation of MIMO instruction-set extensions to implement in hardware. Since MIMO instructions are combinations of MISO instructions and the combination of convex instructions is not always convex, we have to provide a way to combine convex instructions guaranteeing the convexity property of the final MIMO instruction generated. In [13], [11] we presented the following results (Theorem III.5 and Corollary III.6):

Theorem III.5 *Let G be a DAG and $A, B \subset G$ two MMs. Let $\textsc{Lev}(A) \geq \textsc{Lev}(B)$ be the levels of A and B respectively. Let $C = A \cup B$. If $\textsc{Lev}(A) - \textsc{Lev}(B) \in \{0, 1\}$, then C is a convex MIMO. Moreover C is disconnected if the difference is 0.*

Corollary III.6 *Any combination of MAXMISOs at the same level or at two consecutive levels is a convex MIMO.*

This result can be generalized to SMMs and can be used to design clustering algorithms for the identification of convex MIMO instruction-set extensions to implement in hardware.

In [11], [10], we have proposed two clustering algorithms (based on the previous Theorem III.5) which generate convex MIMO instruction-set extensions. In this paper, we take these two algorithms as starting point to design efficient clustering algorithms. More specifically, the main idea is to modify the generation process of the convex MIMOs for the generation of cluster of bigger size which, in turn, can provide higher speed-up when implemented in hardware since the speed-up is usually proportional to the cluster size. The MOLEN architecture is used to carry out the experiments. Although this architecture does not impose limitations on the total number of inputs and outputs of the generated instructions, in this paper, we introduce also constraints on the total number of operands in the generation process. The scope of introducing these constraints is to assess the quality of the presented algorithms on architecture other than MOLEN.

IV. The Clustering Algorithms

The clustering method presented in this paper, is based on the notion of Archimedean Spiral. A spiral is generated when a point P

[4] In general, f collapses a cluster of nodes of G in a node of $\hat{G}$.

moves with constant speed v on a line which, in turn, rotates around one of his points O with constant angular velocity ω. The point O is called **center of the spiral**. The distance T_d between two consecutive turns is called **turn distance**. For additional information see [12]

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

Definition IV.1 If O is a seed node, a node selected as initial node in the generation of a cluster, a **spiral search** is a clustering search which looks for nodes to cluster starting from the level of the 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 satisfy a predefined property P are included in the cluster.

Figure 1 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.

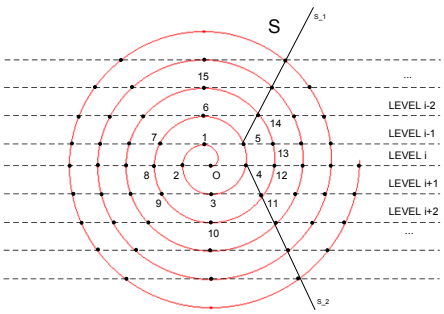

Fig. 1. The spiral search. If $O \in L_i$ is a seed node, the levels are analyzed following the order of intersection between the spiral and the levels: 1, 2, 3, ...

Let G be the DAG of the application and let $\hat{G}$ be the reduced graph. Each cluster is identified starting with a node taken as a seed and, analyzing a certain number of levels, we grow a cluster which includes nodes satisfying specific properties. Let $a_{\overline{\tau}}$ be a node of $\hat{G} = (\hat{V}, \hat{E})$ with $\text{LEV}(a_{\overline{\tau}}) = \alpha \in [0, d]$ and let $C = \{a_{\overline{\tau}}\}$.

The following steps A, ..., D1 are in brief the main steps for the generation of convex MIMO clusters. Let us define the following sets:

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

STEP A. The set:

$$C^I = C \cup \text{PRED}'(a_{\overline{\tau}}) \tag{3}$$

is a convex cluster. This holds for $\alpha \geq 1$ as a consequence of Theorem III.5 and for $\alpha = 0$ since a node is trivially a convex graph.

STEP B. The set:

$$C^{II} = C^I \cup \text{SUCC}'(\text{PRED}'(a_{\overline{\tau}})) \tag{4}$$

is a convex cluster. This holds as a consequence of Theorem III.5.

STEP C1. The set:

$$C^{III} = C^{II} \cup \{m \in \hat{V} \mid \text{LEV}(m) = i + 1 \ \wedge \ N_{In}(m) = N_{In_{C^{II}}}(m)\} \tag{5}$$

is a convex cluster, where $N_{In}(m)$ is the number of inputs of the node m and $N_{In_C}(m)$ is the number of inputs coming from a set C of the node m. The proof is a particular case of the proof in [11].

The algorithm analyzes the nodes of the graph following a spiral search through the levels of the graphs. As described before and depicted in Figure 1, 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 to expand the cluster. We have the following:

Theorem IV.2 *([12, Theorem IV.1]) Intersections 4 and 5 do not provide any node $p \notin C'''$ such that $C^{III} \cup \{p\}$ is convex and $N_{In}(p) = N_{In_{C^{III}}}(p)$.*

Following similar arguments, it is possible to show, see Figure 1, 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.

Let A be a set of nodes with $h \leq \text{LEV}(n) \leq k$ for every node $n \in A$. We can define the following set:

$$\text{PRED}'(A) = \begin{cases} \{m \in V \backslash V_A \mid \text{LEV}(m) = h - 1 \wedge \\ \quad \exists\ (m, n) \in E \text{ with } \text{LEV}(n) = h\} & \text{if } h \geq 1 \\ \emptyset & \text{if } h = 0 \end{cases} \tag{6}$$

STEP D1. By Theorem IV.2 and by the properties of the spiral search, the next level to analyze is level $i - 2$. Let us consider the following sets:

$$C^{IV} = C^{III} \cup \text{PRED}'(C^{III}) \tag{7}$$

$$\begin{aligned} C^V = C^{IV} \cup \{m \in \hat{V} \mid \text{LEV}(m) = i - 1 \wedge \\ \exists (m, a_j) \in \hat{E} \text{ or } (a_j, m) \in \hat{E} \text{ with } a_j \in C^{IV}\} \end{aligned} \tag{8}$$

...

$$\begin{aligned} C^Z = C^{Z-1} \cup \{m \in \hat{V} \mid \text{LEV}(m) = i + 3 \wedge \\ \exists (m, a_j) \in \hat{E} \text{ or } (a_j, m) \in \hat{E} \text{ with } a_j \in C^{Z-1}\} \end{aligned} \tag{9}$$

The cluster C^Z is a convex MIMO cluster (see [10]). In general we have the following:

Remark IV.3 If the level analyzed in STEP D1 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:

$$\text{LEVEL} = i - [(i + \beta) - i] - 1 = i - \beta - 1. \tag{10}$$

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

The algorithm repeats STEP D1 till the final connected convex MIMO cluster is generated. After removing the nodes analyzed from the nodes to further analyze for the generation of other convex MIMO clusters, it restarts from STEP A. This type of analysis implies that the Spiral-clustering requires linear complexity in the number of processed elements.

This algorithm can be modified for the generation of bigger clusters which, in turn, can provide higher speed-up since, in general, the speed-up is proportional to the size of the cluster. In this context, we propose a modified version of algorithm described above, which reiterates STEP A and STEP B. More specifically, after the generation of C^I and C^{II} with STEP A and STEP B respectively, before STEP C1 and D1 we have:

STEP B1. The set:

$$C^{III} = C^{II} \cup \{\text{PRED}'(a_j) \mid a_j \in C^{II} \ \wedge \ \text{LEV}(a_j) = i\} \tag{11}$$

is a convex cluster. This holds as a consequence of Theorem III.5.

STEP B2. The set:

$$C^{IV} = C^{III} \cup \{\text{SUCC}'(a_j) \mid a_j \in C^{III} \ \wedge \ \text{LEV}(a_j) = i - 1\} \tag{12}$$

is a convex cluster. This holds as a consequence of Theorem III.5.

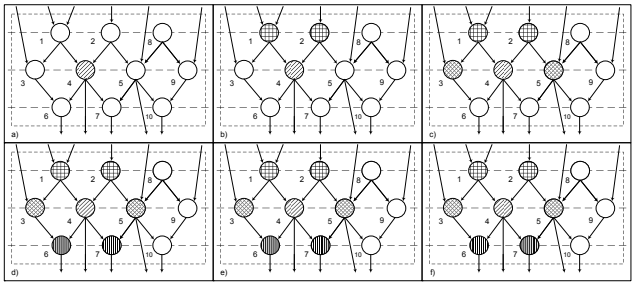

Fig. 2. Clustering without reiteration of STEP A and STEP B: example of an application considering only 3 levels. **a**) $C = \{4\}$, **b**) $C^I = \{4\} \cup \{1,2\}$, **c**) $C^{II} = \{1,2,4\} \cup \{3,5\}$, **d**) $C^{FIN} = \{1,2,3,4,5\} \cup \{6,7\} = \{1,2,3,4,5,6,7\}$, **e**) and **f**) do not expand the cluster.

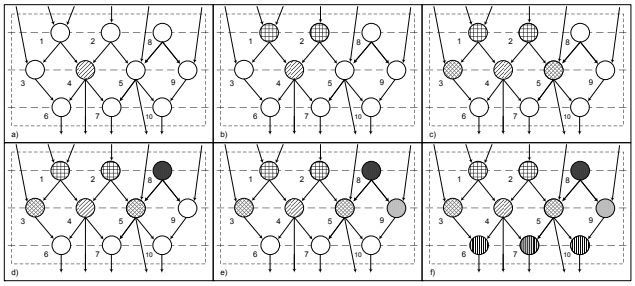

Fig. 3. Clustering with reiteration of STEP A and STEP B: example of an application considering only 3 levels. **a**) $C = \{4\}$, **b**) $C^I = \{4\} \cup \{1,2\}$, **c**) $C^{II} = \{1,2,4\} \cup \{3,5\}$, **d**) $C^{III} = \{1,2,3,4,5\} \cup \{8\}$, **e**) $C^{IV} = \{1,2,3,4,5,8\} \cup \{9\}$, $C^{FIN} = \{1,2,3,4,5,8,9\} \cup \{6,7,10\} = \{1,2,3,4,5,6,7,8,9,10\}$.

After that, the algorithm continues with STEP C1 and STEP D1 (C^{III} in STEP C1 becomes now C^{IV} and so on).

Figure 2 and 3 depict an easy example which shows the reiteration and non-reiteration of STEP A and B.

Additionally, if in place of STEP D1, we reiterate STEP C1 increasing one level at each iteration ($i \rightarrow i+1$) until there are nodes that can be included in the cluster, the algorithm generates convex MIMO clusters as in [11] with or without reiteration of the initial steps (i.e. with or without STEP B1 and B2).

Concerning the computational complexity, the algorithms take as an input a node of $\hat{V}$, generate the cluster C^{FIN} and remove the nodes belonging to the cluster from the node to further analyze for identifying other convex MIMO clusters. In this way each node is analyzed only once keeping the overall computational complexity of the algorithms linear with the number of processed elements.

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

a) Selection of an application;
b) MM partitioning: the application is analyzed and partitioned in MMs using an algorithm similar to the one presented in [2];
c) SMM partitioning: the application partitioned in MM is sub-partitioned in SMMs using an algorithm similar to the one presented in [9] and considering different options for the selection of the the node removed from the MMs (see Section V);
d) Convex MIMO Clustering: the application partitioned in SMMs is analyzed and convex MIMO clusters are built following step A, B, (B1, B2) C1, D1 or A, B, (B1, B2) C1, C1, ...

As mentioned before, in this paper, we introduce limitations on the total number on inputs and outputs of the convex clusters generated by the algorithms. Since the clusters are generated in multiple steps

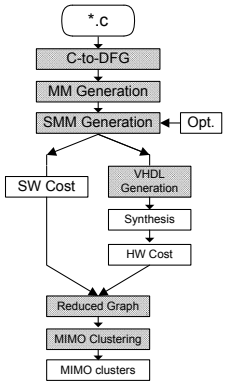

Fig. 4. Tool Chain for the identification of instruction-set extensions.

and at each step additional nodes are included in the cluster, the number of inputs and outputs of the clusters can increase or decrease at each step. Stop the clustering if the input/output constraints is violated can produce extremely small clusters. At present, the algorithms complete the cluster generation without introducing constraints on inputs and/or output during the clustering. If limitation on inputs and/or outputs are introduced, the algorithms return the last convex cluster built during the generation process which satisfy input and output constraints. In this way there are more chances to have a bigger cluster.

V. Experimental Setup and Results

A. Experiments setting

To evaluate the qualities of the presented clustering algorithms, a dedicated tool chain has been built and the algorithm has been applied on a set of well-known kernels. The presented clustering algorithms are part of a larger toolchain that aims at supporting the hardware designer in the design process (see also [10], [23] for more details). The tool chain for the experiments is presented in Figure 4 The input is C code in which the kernel functions are marked with pragmas. The annotated functions are transformed into dataflow graphs (DFGs). The generated graphs are analyzed for the MM and SMM partitioning.

The software execution time for each SMM is computed as the sum of the latencies of its operations. The hardware execution time is estimated through behavioral synthesis of the MM and SMM's VHDL models using the Xilinx ISE 9.4i, and this delay is converted into PowerPC cycles. We consider implementation of our approach on the Molen prototype that is built on a Xilinx Virtex-II Pro XC2VP100 FPGA device. The software execution is assumed to be performed on a PowerPC 405 operating at 300MHz. VHDL synthesis is performed for the target board to estimate area and delay of the MM and SMMs. Since the PowerPC processor does not provide floating-point instructions, the floating-point operations in the benchmark kernels are converted into the proper integer arithmetic. The kernels have been unrolled by a factor of 8/16 in order to increase the selection space of our algorithms. *The tools in the toolchain do not require any manual effort for adjusting to the target application.*

The estimated software and hardware implementation costs are used by the coarse-grain clustering algorithms for the generation of convex MIMO instructions. The algorithms select a node in the graph as a seed for the growth of the final cluster and iteratively group nodes following a certain number of steps as previously described in Section IV.

The clustering is limited by the size of the reconfigurable hardware. This means that the algorithm stops the generation of convex MIMOs

if there exists no extra design space to explore for clustering or if there is no more available area in the FPGA. In the second case, if the total area of the generated clusters exceeds the available area on the reconfigurable hardware, we use an established metric for evaluating our optimizations and select the instructions to implement on the reconfigurable hardware, which fit the available area. The Area-Delay Product (ADP) and its variants (e.g. A2DP, AD2P) have been used extensively [16], [14] and provide a well-known empirical measure of performance. Delay is the primary concern in our working context; thus $AD2P$ has been used as our metric. In effect, the clusters are ranked based on the following formula: $A_C/(L_{SW} - L_{HW})^2$, where A_C is the area of the generated cluster, L_{SW} and L_{HW} are the latency of the cluster in software and hardware respectively.

B. Analysis of the data

Given an application, it is initially partitioned in MMs. After that, the SMM clustering selects a node in each MM and applies the MM clustering to the MMs bereft of the selected nodes. This makes the SMM clustering node-dependent. For this reason, seven versions of the FIX SMM clustering algorithm have been designed and implemented depending on the node removed in the MMs, as described in the following:

- Option 1: an input node.
- Option 2: an output node.
- Option 3: a random node.
- Option 4: 1^{st} node with 1 successor and 1 predecessor.
- Option 5: 1^{st} node with 1 successor and 2 predecessors.
- Option 6: 2^{nd} node with 1 successor and 1 predecessors.
- Option 7: 2^{nd} node with 1 successor and 2 predecessors.

Since the DFG of the application is a DAG, the nodes of the graph can be topologically ordered. Successors and predecessors of nodes mentioned before are then identified by the topological order of the nodes. The node to remove is chosen starting the analysis from the output node of the MM. This means for example, that in option 3 the output node of each MM is removed and the MM partitioning is applied to each MM bereft of its output. Concerning the selection of the seed in the coarse-grain clustering, the node is selected as the one with smallest latency in hardware. In case many nodes have the same latency, the seed is randomly selected starting from the lowest levels of the reduced graph.

In our experiments, we have used well known cryptographic kernels from MCRYPT library (http://mcrypt.sourceforge.net/), the Adaptive Differential Pulse-Code Modulation decoder (ADPCM, a well-known audio format decoder) and the Sum of Absolute Differences (SAD).

In Figure 5, we depict the overall application speed-up for the Virtex II PRO XC2VP100 FPGA device compared to the pure software execution for the different kernels.

An obvious remark concerns Option 1 of the SMM clustering. As Figure 5 depicts, in most of the cases, Option 1 does not perform well. This is the result of removing an input node from a MM. It generates only two SMMs: the node removed and the MM bereft of the selected nodes. The limited number of SMMs available to combine compared with the number of SMMs generated by the other version of the SMM clustering affects the number of possible combinations. On average, the best choices in the selection of the node to remove for the generation of the SMMs are option 4–7 which remove an inner node from the clusters to partition and generate bigger clusters which, when implemented in hardware, provide higher speed-up. This, in turn, leads to the generation of a big number of SMMs which allows for a finer selection of the final cluster.

Concerning the overall computational complexity of the clustering algorithms presented in this paper, it is the sum of the complexities of the single-output partitioning and of the multiple-output clustering. The MM (SMM) partitioning and the algorithms presented in this paper, both require linear complexity in the number of processed elements. As a consequence, the overall complexity of the clustering algorithms is linear in the number of processed elements.

Concerning the area usage on the FPGA, ADPCM Decoder and TWOFISH Encoder use between 20% and 30% of area and CAST 128 Encoder and SAD use between 10% and 20% of area. Hardware reuse is not considered in our approach. This because the inclusion of hardware reuse in the generation process turns into an analysis of the generated clusters to identify isomorphic graphs and avoid the implementation in hardware of the same instructions. Since we want to keep an overall linear complexity and the isomorphism problem is a well known computationally complex problem, hardware reuse has not been considered up to now. This turns in additional hardware usage. Future research will included also this problem in the generation process.

Unfortunately, the benefit of the inclusion of our clustering algorithm for automatic ISE can not be compared exhaustively with the work presented in literature. This is mainly due to the lack in literature of a basic set of benchmark applications and kernels used to test the various methodologies. Comparing with the work presented by [4], which provides an optimal solution, our overall speed-up for the ADPCM decoder is x2.2 with an overall linear complexity in contrast with the x3.4 and an exponential overall complexity of the approach of [4]. Additionally, our algorithms do not have problems of scalability. In [3], despite the different benchmarks, we both use cryptographic benchmarks and the overall application speed-up has the same order of magnitude (x5.5 with our approach and x7.5 with the approach presented in [3] but which still has high overall computational complexity in contrast with the linear complexity of our approach. Compared with the work presented in [11] (Figure 5), the clustering algorithms presented in this paper perform better in most of the cases.

VI. Conclusions

In this paper, we have introduced general algorithms for the automatic identification of convex MIMO instruction-set extensions. The proposed clustering algorithms combine clusters of single-output instructions of variable size (SMMs) for execution as new application-specific MIMO instructions on the reconfigurable hardware. The proposed clusterings generate convex MIMO instructions with linear complexity in the number of processed elements and do not impose limitations either on the types and number or on the number of inputs and outputs of the generated instructions. Our results show that a significant overall application speed-up is achieved even for large kernel (for ADPCM decoder the speed-up is up to x2.2 and for TWOFISH encoder the speed-up is up to x5.5).

Acknowledgment

This work was supported by the European Union in the context of the MORPHEUS project (IST-027342), the hArtes project (IST-035143) and RCOSY project (DES-6392).

References

[1] MOLEN Reconfigurable Processors – http://ce.et.tudelft.nl/MOLEN/.
[2] C. Alippi et al. A dag-based design approach for reconfigurable vliw processors. In *DATE '99*
[3] K. Atasu et al. An integer linear programming approach for identifying instruction-set extensions. In *CODES+ISSS '05*,

a) ADPCM Decoder 1UD(1) **b) CAST 128 Decoder 1UD(1)** **c) SAD 1UD(1)**

d) ADPCM Decoder SPIR(1) **e) CAST 128 Decoder SPIR(1)** **f) SAD SPIR(1)**

g) TWOFISH Encoder 1UD(1) **h) TWOFISH Encoder SPIR(1)**

Fig. 5. Overall speed-up for different kernels and different clustering algorithms. Two algorithms have been tested: 1UD(1), and SPIR(1) where SPIR(1) represent the clustering through the spiral search with reiteration of the initial steps and 1UD(1) is the modified version of the clustering algorithm presented in [11] with reiteration of the initial steps. For each algorithm different fine-grain partitioning are considered: the MM partitioning and the SMM partitioning with different selection in the node removed for the generation of the SMMs (Opt. 1–7). The algorithms are tested including limitations on the total number of inputs and outputs. The speed-up estimation is based on Amdahl's law, using the profiling results and the computed speed-up for the kernels.

[4] K. Atasu et al. Automatic application-specific instruction-set extensions under microarchitectural constraints. In *DAC '03*

[5] M. Baleani et al. Hw/sw partitioning and code generation of embedded control applications on a reconfigurable architecture platform. In *CODES '02*

[6] P. Biswas et al. Isegen: Generation of high-quality instruction set extensions by iterative improvement. In *DATE '05*

[7] N. Clark et al. Processor acceleration through automated instruction set customization. In *MICRO 36*

[8] J. Cong et al. Application-specific instruction generation for configurable processor architectures. In *FPGA '04*

[9] C. Galuzzi et al. A linear complexity algorithm for the generation of multiple input single output instructions of variable size. SAMOS VII

[10] C. Galuzzi et al. Automatic Instruction-Set Extensions with the Linear Complexity Spiral Search. ReConFig 2008

[11] C. Galuzzi et al. A linear complexity algorithm for the automatic generation of convex multiple input multiple output instructions. In ARC07

[12] C. Galuzzi et al. The Spiral Search: A Linear Complexity Algorithm for the Generation of Convex MIMO Instruction-Set Extensions. In FPT07

[13] C. Galuzzi et al. Automatic selection of application-specific instruction-set extensions. In *CODES+ISSS '06*

[14] A. Gayasen et al. Switch box architectures for three-dimensional fpgas. In *FCCM '06*

[15] R. Kastner et al. Instruction generation for hybrid reconfigurable systems. *ACM TODAES*, 7(4):605–627, 2002.

[16] I. Kuon and J. Rose. Area and delay trade-offs in the circuit and architecture design of fpgas. In *FPGA '08*

[17] C. Lee et al. Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems. In *MICRO 30*

[18] R. Leupers et al. A design flow for configurable embedded processors based on optimized instruction set extension synthesis. In *DATE '06*

[19] L. Pozzi and P. Ienne. Exploiting pipelining to relax register-file port constraints of instruction-set extensions. In *CASES '05*

[20] F. Sun et al. Synthesis of custom processors based on extensible platforms. In *ICCAD '02*

[21] S. Vassiliadis et al. The molen polymorphic processor. *IEEE ToC*, 53(11):1363–1375, 2004.

[22] C. Wolinski and K. Kuchcinski. Automatic selection of application-specific reconfigurable processor extensions. In *DATE '08*

[23] Y. Yankova et al. DWARV: Delftworkbench automated reconfigurable vhdl generator. In *FPL'07*

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

[25] P. Yu and T. Mitra. Disjoint pattern enumeration for custom instructions identification. In *FPL 2007*