A Linear Complexity Algorithm for the Automatic Generation of Convex Multiple Input Multiple Output Instructions

Carlo Galuzzi, Koen Bertels, and Stamatis Vassiliadis

Computer Engineering, EEMCS
Delft University of Technology, The Netherlands
{C.Galuzzi, K.L.M.Bertels, S.Vassiliadis}@ewi.tudelft.nl

Abstract. The Instruction-Set Extensions problem has been one of the major topic in the last years 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. This search turns into an exponential complexity of the solution. In this paper we propose an efficient linear complexity algorithm for the automatic generation of convex Multiple Input Multiple Output (MIMO) instructions, whose convexity is theoretically guaranteed. The proposed approach is not restricted to basic-block level and does not impose limitations either on the number of input and/or output, or on the number of new instructions generated. Our results show a significant overall application speedup (up to x2.9 for ADPCM decoder) considering the linear complexity of the proposed solution and which therefore compares well with other state-of-art algorithms for automatic instruction set extensions.

1 Introduction

The last years have shown 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. The execution of an application on an architecture with a given Instruction-Set usually involves instructions belonging to the Instruction-Set, and implemented in hardware, and many instructions executed in software, and typically more costly in terms of execution time than the ones executed in hardware. Roughly speaking the idea is to group clusters of instructions executed in software in new complex instructions to implement on the reconfigurable hardware and whose execution time is faster in hardware than in software. Once these new complex instructions are hardwired, they represent

P.C. Diniz et al. (Eds.): ARC 2007, LNCS 4419, pp. 130–141, 2007.

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.

In this paper we propose a linear complexity algorithm for the automatic generation of new Application-Specific Instructions under hardware resources constraints. The proposed approach targets the Molen organization [1] 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.

A set of convex Multiple Input Multiple Output (MIMO) operations is identified by a two steps approach. Firstly, the operations are clustered in Maximal Multiple Input Single Output (MAXMISO) operations, the elementary building blocks of our approach, and subsequently these MAXMISOs are clustered per levels as new application-specific instructions. The result is a cluster of operations with Multiple Input Multiple Output, called MIMO, which is executed on the reconfigurable hardware and which provides the maximum performance improvement under reconfigurable hardware resource constraints. More specifically, the main contributions of this paper are:

- an overall linear complexity of the proposed solution. 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 approach heuristically extends a given Instruction-Set with convex MIMO instructions based on MAXMISOs clustering in order to exploit the MAXMISO level parallelism. Single MAXMISOs usually do not provide significant performance improvement. Thus we propose MAXMISOs combination (Section 3.4) in order to take advantage of the parallelism inherent to the hardware execution and the Theorem 2 that guarantees the MIMO convexity by construction. The proposed solution addresses the problem with two linear complexity steps, MAXMISO clustering and MAXMISO combination, that are linear in the number of processed elements.
- 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 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 organized as follows. In Section 2, background information and related works are provided. In Section 3, the context is further formalized and the theoretical contribution is presented. Section 4 presents the experimental setup and results. Concluding remarks and an outline of research conducted are given in Section 5.

2 Background and Related Works

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

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

Concerning the first category, a representative example is introduced in [2] which addresses the generation of MISO instructions of maximal size, called MAXMISO. The proposed algorithm exhaustively enumerates all MAXMISOs. Its complexity is linear with the number of nodes. The reported performance improvement is of few processor cycles per newly added instruction. The approach presented in [3] targets the generation of general MISO instructions. The exponential number of candidate instructions turns into an exponential complexity of the solution in the general case. As a consequence, heuristic and additional area constraints are introduced to allow an efficient generation. The difference between the complexity of the two approaches is due to the properties of MISOs and MAXMISOs: while the enumeration of the first is similar to the subgraph enumeration problem (which is exponential) the intersection of MAXMISOs is empty and then once a MAXMISO is identified, its nodes are removed from the set of nodes that have to be successively analyzed. In this way the MAXMISOs are enumerated with linear complexity in the number of nodes.

The algorithms included in the second category are more general and provide more significant performance improvement. However, they have exponential complexity. For example, in [4] the identification algorithm detects optimal convex MIMO subgraphs but the computational complexity is exponential. A similar approach described in [5] proposes the enumeration of all the instructions based on the number of inputs, outputs, area and convexity. The selection problem is not addressed. In [6] the authors target the identification of convex clusters of operations under given input and output constraints. The clusters are identified with a ILP based methodology similar to the one proposed in [7]. The main difference is that in [6] the authors iteratively solve ILP problems for each basic block, while in [7] the authors have one global ILP problem for the entire procedure. Additionally, the convexity is addressed differently: in [6], the convexity is verified at each iteration, while in [7] it is guaranteed by construction. Other approaches cluster operations by considering the frequency of execution or the occurrence of specific nodes [8,9] or regularity [10]. Still others impose limitation on the number of operands [11,12] and use heuristics to generate sets of custom instructions which therefore can not be globally optimal.

The algorithm we introduce in this paper combines concepts of both categories: at first, MAXMISOs are identified as proposed in the first category, afterward they are combined in convex MIMOs. Our algorithm requires linear complexity for the MAXMISO enumeration and the MAXMISO combination. Additionally, the proposed algorithm does not impose any limitations on the number of input/output values (as in [13,14,11]) or on the number of newly added instructions.

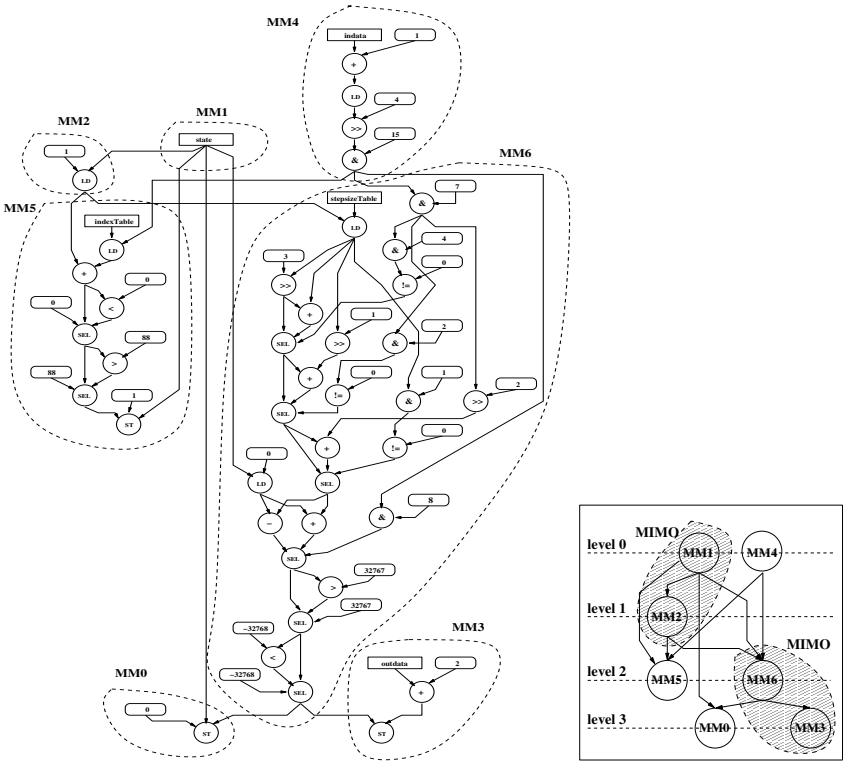

Fig. 1. Motivational example: Dataflow subgraph from *ADPCM* Decoder. **a)** the MAXMISO identification and **b)** the collapsed graph with the convex MIMOs identified by the algorithm.

3 Theoretical Background

In this section, we begin by introducing a motivational example to informally outline the main concept of the proposed algorithm and then we present the theoretical foundation of our approach. Last but not least, we present in detail the steps of the MIMO clustering algorithm.

3.1 Motivational Example

In Figure 1, we present the dataflow subgraph of the ADPCM application as implemented in the MediaBench benchmark suite [15]. Our algorithm identifies the convex MIMO in two steps. Firstly, the graph is partitioned in MAXMISOs (see Figure 1(**a**)). Each MAXMISO is then collapsed as a single node in the reduced graph presented in Figure 1(**b**). Since single MAXMISO execution in hardware doesn't provide significant performance improvement, the main idea of our algorithm is to combine, per levels, MAXMISOs available at the same

level in the reduced graph, into a convex MIMO that is executed as a single instruction in hardware. The combination per level and successively per levels is the key difference between the solution presented in this paper and the one we proposed in [7]. Let assume the hardware latency for $MAXMISO_i$ to be l_i. When k MAXMISOs at the same level are clustered, the execution time of the cluster in hardware is $\max_{i=1..k} l_i$. The performance gain in this case is $\sum_{i=1..k}(l_i) - \max_{i=1..k}(l_i)$. If successively we cluster per levels, the overall performance gain increases. Let assume that $\alpha_1, ..., \alpha_h$ are the levels of the nodes belonging to a cluster generated combining per levels. The overall performance gain in this case is:

$$\sum_{j=\alpha_1}^{\alpha_h} (\sum_{i_j} l_{i_j} - \max_{i_j}(l_{i_j})) \tag{1}$$

Although the clustering algorithm is detailed in the following sections, we can roughly underline the main clustering steps. Starting from a node v belonging to level m, the algorithm grows a convex cluster of nodes C taking v as a seed. At first, an initial cluster C' composed by v and its predecessors at level $m-1$ is grown. C' is further extended with the successors at level m of the nodes of $C' \backslash \{v\}$. By Theorem 2, this extension C'' is convex. Once C'' is generated, the algorithm iteratively extends at each iterations C'' with nodes having all inputs coming from nodes belonging to the cluster generated in the previous iteration. When the cluster is no more extendable, its nodes are removed from the nodes to analyze and the algorithm restarts a new cluster from a node v'. The final clusters $C_1, .., C_l$ built in this way are convex MIMOs (see Section 3.4). For the graph presented in Figure 1(**a**) the algorithm identifies the convex clusters $C_1 = \{MM_1, MM_2\}$ and $C_2 = \{MM_3, MM_6\}$.

MAXMISO clustering is limited by the size of the reconfigurable hardware. This means that the algorithm stops the generation of convex MIMO as soon as there is no more available area in the FPGA.

3.2 MISO Properties and MAXMISO-Clustering

In order to formally express the problem 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.

Let $\wp(G)$ be the power set of G[1] and let n be the order of V. The order of $\wp(G)$ is 2^n. Basically, there are two types of subgraphs that can be identified: MISOs and MIMOs. 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:

$$G_{MISO} \subset G_{MIMO} \subset \wp(G). \tag{2}$$

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

Definition 1. *Let $G^* \subseteq G$ be a subgraph of G with $V^* \subseteq V$ set of nodes and $E^* \subseteq E$ set of edges. G^* is a MISO of root $r \in V^*$ provided that $\forall\, v_i \in V^*$ there exists a path[2] $[v_i \rightarrow r]$, and every path $[v_i \rightarrow r]$ is entirely contained in G^*.*

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

Definition 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 1 and 2 imply that every MISO is a connected and convex graph. MIMOs can be convex or not. For example the subgraph $G^* = \{MM_1, MM_6\}$ in Figure 1(**b**) is not a convex graph.

Nevertheless the following property holds.

Theorem 1. *Let $G^* \subsetneq G$ be a convex subgraph of G. Then there exists $k \in \mathbb{N}$ such that $G^* = \bigcup_{i=1}^{k} MISO_i$.*

Proof. Let G^* be a MISO. Then $k = 1$ and $MISO_1 = G^*$. Let G^* be a MIMO. Every node has single output 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 implies that an exhaustive enumeration of the MISOs contained in G gives all the necessary building blocks to generate all possible convex MIMOs. This faces with the exponential order of G_{MISO} and, by (2), of G_{MIMO}. 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.

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 MAXMISO can formally be defined as follows.

Definition 3. *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.*

We know from the set-theory that each element of G_{MISO} is either maximal (a MAXMISO) or there exists a maximal element containing it. [2] 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:

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

[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.

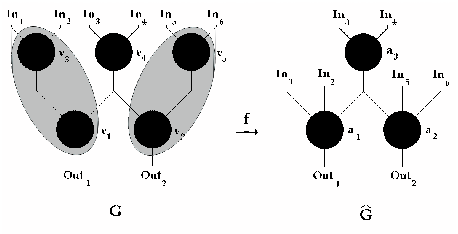

Fig. 2. The collapsing function $f : G \to \hat{G}$. $G_{MM} = \{MM_1, MM_2, MM_3\}$ where $MM_1 = \{v_1, v_3\}, MM_2 = \{v_4\}, MM_3 = \{v_2, v_5\}$. Then $MM_1 \mapsto a_1$, $MM_2 \mapsto a_3$ and $MM_3 \mapsto a_2$. In this case we have $G_{MISO} = \{v_1, v_2, v_3, v_4, v_5, MM_1, MM_3\}$ and clearly $G_{MM} \subsetneq G_{MISO}$.

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.

3.3 Convex MIMOs Generation

Let $G_{MM} \subset G_{MISO}$ be the set of all MAXMISOs of G. Let $f : G_{MM} \subset G \to \hat{G}$ be the function that collapses the MAXMISOs of G in nodes of the graph $\hat{G}$, (Fig. 2): $MM_i \in G_{MM} \subset G \mapsto a_i \in \hat{G}$.

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

Let $v \in V$ be a node of G and let $\text{LEV} : V \to \mathbb{N}$ be the integer function defined as follows:

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

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

Definition 4. *The level of $MAXMISO_i \in G$ is defined as follows:*
$$\text{LEV}(MAXMISO_i) = \text{LEV}(f(MAXMISO_i)).$$

Theorem 2. *Let G be a DAG and $A_1, A_2 \subset G$ two MAXMISOs*[4]*. Let $\text{LEV}(A_1) \geq \text{LEV}(A_2)$ be the levels of A_1 and A_2 respectively. Let $C = A_1 \cup A_2$. If*

$$\text{LEV}(A_1) - \text{LEV}(A_2) \in \{0, 1\} \tag{4}$$

then C is a convex MIMO. Moreover C is disconnected if the difference is 0.[5]

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

[5] $\text{LEV}(A_1) - \text{LEV}(A_2) = 0$ is a particular case studied in [7].

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 (4) 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

$$\text{LEV}(a_2) < \text{LEV}(a_k) < \text{LEV}(a_1). \tag{5}$$

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 hypothesis $\text{LEV}(\cdot) \in \mathbb{N}$. 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 contradicts 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 1. *Any combination of MAXMISOs at the same level or at two 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: $\text{LEV}(A_i) = \overline{\iota} \ \ \forall i$, or $\text{LEV}(A_i) \in \{\overline{\iota}, \overline{\iota}+1\}$. 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 2. Then c is a convex MIMO as well as $C = f^{-1}(c)$. □

3.4 The 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 further extended with additional nodes.

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

$$\begin{aligned}
\text{PRED}'(a_{\overline{\iota}}) &= \begin{cases} \{m \in \hat{V} \mid \text{LEV}(m) = \alpha - 1 \ \wedge \ \exists\, (m, a_{\overline{\iota}}) \in \hat{E}\} & \text{if } \alpha \geq 1 \\ \emptyset & \text{if } \alpha = 0 \end{cases} \\
\text{SUCC}'(a_{\overline{\iota}}) &= \begin{cases} \{m \in \hat{V} \mid \text{LEV}(m) = \alpha + 1 \ \wedge \ \exists\, (a_{\overline{\iota}}, m) \in \hat{E}\} & \text{if } \alpha \leq d - 1 \\ \emptyset & \text{if } \alpha = d \end{cases} \\
\text{SUCC}(a_{\overline{\iota}}) \ &= \begin{cases} \{m \in \hat{V} \mid \exists\, [a_{\overline{\iota}} \rightarrow m] \wedge \ \text{LEV}(m) > \text{LEV}(a_{\overline{\iota}})\} & \text{if } \alpha \leq d - 1 \\ \emptyset & \text{if } \alpha = d. \end{cases}
\end{aligned} \tag{6}$$

$C' = C \ \cup \ \text{PRED}'(a_{\overline{\iota}})$ is a convex MIMO. This holds for $\alpha \geq 1$ as a consequence of Theorem 2 and for $\alpha = 0$ since a node is trivially a convex graph.

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:

$$2 * N_{In_C}(n) \geq N_{In}(n). \tag{7}$$

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

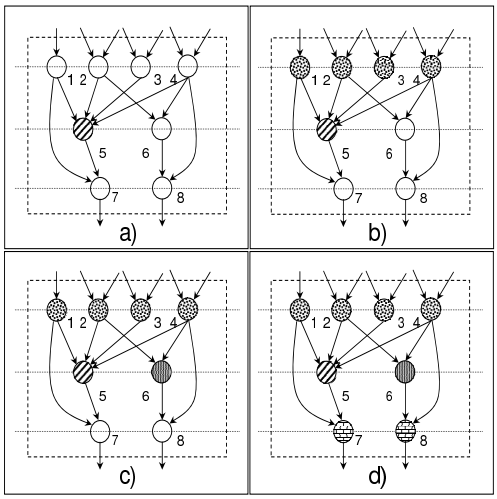

Fig. 3. Example of an application of the algorithm. **a**) $C = \{5\}$, **b**) $C' = \{5\} \cup \{1,2,3,4\}$, **c**) $C'' = \{1,..,6\}$, **d**) $C''' = \{1,..,8\}$.

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:

$$C'' = \begin{cases} C' \cup \{n \in \text{SUCC}'(\text{PRED}'(a_{\overline{\tau}})) \mid (7) \text{ holds}\} & \text{if } n \text{ exists} \\ C' & \text{otherwise.} \end{cases} \tag{8}$$

If there exists n such that (7) holds, by Theorem 2, $C'' = C' \cup \{n\}$ is a convex MIMO.

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

$$\text{SUCC}(C'') = \bigcup_{m \in C''} \text{SUCC}(m). \tag{9}$$

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

$$N_{In}(m) = N_{In_{C''}}(m), \tag{10}$$

$C'' \cup \{m\}$ is a convex MIMO. This follows from (10). If the total number of inputs of m is equal to the number of inputs coming from C'', it follows that it doesn't 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 $C''' = C'' \cup \{m \in \text{SUCC}(C'') \mid N_{In_C}(m) = N_{In}(m)\}$ is a convex MIMO. In Figure 3, we present an example that shows the way by which the algorithm generates the convex MIMO C'''.

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 [2]
- 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 $\hat{G}$, generate C''' and remove the nodes of C''' from the node to further analyze.

Remark 1. By Step d, it follows that the convex MIMOs are generated linearly with the number of node of $\hat{G}$. Additionally, the inequality (7) is introduced to limit the total number of inputs of C''. The node which is selected to be the seed to grow C''' is chosen as the one with smallest latency in hardware. In case many nodes have the same latency, at present we select randomly the seed starting from the lowest levels.

4 Results

To evaluate the speedup achieved by the proposed approach, a dedicated tool chain is built and the algorithm is applied on a set of four well-known kernels and benchmarks applications. The above described algorithm is part of a larger automatic tool chain that aims to support the hardware designer in the design process. The tool chain for the experiments has been already described in our previous paper [7].

The software execution time for each MAXMISO is computed as the sum of the latencies of its operations. The hardware execution time is estimated through behavioral synthesis of the MAXMISO's VHDL models and then converting the reported delay into PowerPC cycles. We consider implementation of our approach on the Molen prototype that is built on Xilinx's Virtex-II Pro Platform FPGA. Since the PowerPC processor does not provide floating-point instructions, the floating-point operations in the benchmark suite kernels are converted into the proper integer arithmetic. The DCT, IDCT, and ADPCM decoder kernels have been unrolled by a factor of 8/16 in order to increase the selection space of our algorithm. In the current MOLEN prototype, the access to the Exchange Registers (used for GPP-FPGA communication) (XRs) for the input/output values is significantly slower compared to the GPP register. As this is a limitation only in the current prototype and taking into account that other approaches on Instruction-Set Extension do not consider register accesses, for a fair comparison we report two set of results: with and without XR accesses. For our experiments, we consider a set of three well-known MediaBench [15] benchmarks and MJPEG encoder application. In Figure 4, we present the overall application speedup for FPGAs of various sizes compared to the pure software execution.[7] For ADPCM and SAD we only presented a small set of FPGAs since a further increase of the FPGA size does not provide additional improvement. The speedup estimation is

[7] The occupied area is not shown, since it is almost equal to the available area on the FPGAs.

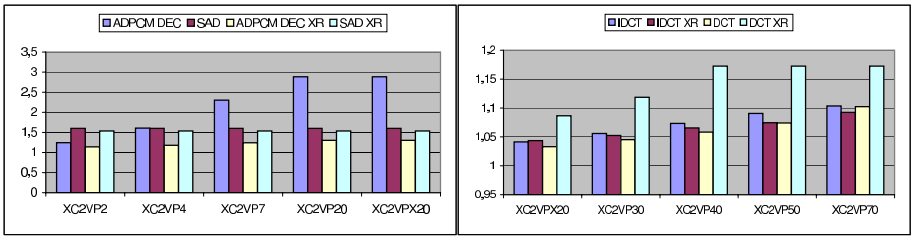

Fig. 4. Overall speedup for different FPGA sizes of **a**) ADPCM Decoder/MPEG2 Encoder, and **b**) MJPEG Encoder/MPEG2 Decoder

based on Amdahl's law, using the profiling results and the computed speedup for the kernels. The achieved speedup varies from x1.2 up to x2.9 for the ADPCM Decoder and different FPGA sizes. For the other benchmarks, the speedup is limited due to the shape of the dataflow graphs of the application.

An expected observation is that the impact on performance of the MM-Level algorithm increases with the size of the available FPGA area. This is explained by the fact that more MAXMISOs can be combined for hardware execution on the FPGAs. Additionally, we notice that for some applications/kernels (MPEG2 Encoder/SAD) the estimated speedup does not depend on the FPGA size. This is due to the fact that SAD kernel contains only one MAXMISO which fits on all FPGAs. A second observation is that the impact of the XR accesses on the overall speedup is higher for the ADPCM decoder compared to the SAD/DCT/IDCT kernels. This is due to the larger number of input/output values of the selected MAXMISOs and high XR access latency. A last observation is about the shape of the graph. The algorithm we propose in this paper is designed to work properly with graph having a large depth. The benchmarks we proposed to test our algorithm are usually represented by wide graphs with a small depth which limits the extension of C'' in C'''. Promising benchmarks to test our algorithm are, for example, the cryptographic benchmarks which usually are applications represented by graph with a large depth but at present they are not ready to test the algorithm.

It is important to note at this point that even though the overall speedup is limited, the overall complexity of our solution is linear in the number of processed elements[8]. Moreover, we emphasize that our approach does not impose limitations on the number of operands and on the number of new instructions.

5 Conclusions

In this paper, we have introduced an algorithm which combines clusters of MAXMISO for execution as new application-specific instructions on the reconfigurable

[8] More specifically, the number of processed elements is at most $n+m$, where n is the order of G and m is the number of MAXMISOs in G.

hardware. One of the main features of our approach is the linear overall complexity. Additionally, the proposed algorithm is general: new instructions have no limitation on their types and numbers and the algorithm is applied beyond basic block level. The used SW model in the experiments is simplified and does not reflect the available processor optimizations (pipelines, cache hits, etc). Besides we do not consider also the possible penalties like branch miss-predictions, cache misses, etc. Even though the generated VHDL is not optimized, we consider our results reliable and promising for future studies. A more general MAXMISO-clustering and operation-clustering concern a task for our future research plan.

References

1. Vassiliadis, S., Wong, S., Gaydadjiev, G., Bertels, K., Kuzmanov, G., Panainte, E.M.: The molen polymorphic processor. IEEE Trans. Comput. **53**(11) (2004) 1363–1375
2. Alippi, C., Fornaciari, W., Pozzi, L., Sami, M.: A dag-based design approach for reconfigurable vliw processors. (In: Proceedings of DATE '99)
3. Cong, J., Fan, Y., Han, G., Zhang, Z.: Application-specific instruction generation for configurable processor architectures. (In: Proceedings of FPGA '04)
4. Atasu, K., Pozzi, L., Ienne, P.: Automatic application-specific instruction-set extensions under microarchitectural constraints. (In: Proceedings of DAC '03)
5. Yu, P., Mitra, T.: Scalable custom instructions identification for instruction-set extensible processors. (In: Proceedings of CASES '04)
6. Atasu, K., Dündar, G., Özturan, C.: An integer linear programming approach for identifying instruction-set extensions. (In: Proceedings of CODES+ISSS '05)
7. Galuzzi, C., Panainte, E.M., Yankova, Y., Bertels, K., Vassiliadis, S.: Automatic selection of application-specific instruction-set extensions. (In: Proceedings of CODES+ISSS '06)
8. Kastner, R., Kaplan, A., Memik, S.O., Bozorgzadeh, E.: Instruction generation for hybrid reconfigurable systems. ACM Trans. Des. Autom. Electron. Syst. **7**(4) (2002) 605–627
9. Sun, F., Ravi, S., Raghunathan, A., Jha, N.K.: Synthesis of custom processors based on extensible platforms. (In: Proceedings of ICCAD '02)
10. Brisk, P., Kaplan, A., Kastner, R., Sarrafzadeh, M.: Instruction generation and regularity extraction for reconfigurable processors. (In: Proceedings of CASES '02)
11. Baleani, M., Gennari, F., Jiang, Y., Patel, Y., Brayton, R.K., Sangiovanni-Vincentelli, A.: Hw/sw partitioning and code generation of embedded control applications on a reconfigurable architecture platform. (In: Proceedings of CODES '02)
12. Clark, N., Zhong, H., Mahlke, S.: Processor acceleration through automated instruction set customization. In: Proceedings of MICRO 36. (2003) 129
13. Goodwin, D., Petkov, D.: Automatic generation of application specific processors. (In: Proceedings of CASES '03)
14. Choi, H., Hwang, S.H., Kyung, C.M., Park, I.C.: Synthesis of application specific instructions for embedded dsp software. (In: Proceedings of ICCAD '98)
15. Lee, C., Potkonjak, M., Mangione-Smith, W.H.: Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems. In: Proceedings of MICRO 30. (1997) 330–335