Automatic Selection of Application-Specific Instruction-Set Extensions

Carlo Galuzzi, Elena Moscu Panainte, Yana Yankova, Koen Bertels, and Stamatis Vassiliadis
Computer Engineering, EEMCS
Delft University of Technology
Delft, The Netherlands
{C.Galuzzi, E.Panainte, Y.D.Yankova, K.L.M.Bertels, S.Vassiliadis}@ewi.tudelft.nl

ABSTRACT

In this paper, we present a general and an efficient algorithm for automatic selection of new application-specific instructions under hardware resources constraints. The instruction selection is formulated as an ILP problem and efficient solvers can be used for finding the optimal solution. An important feature of our algorithm is that it is not restricted to basic-block level nor does it impose any limitation on the number of the newly added instructions or on the number of the inputs/outputs of these instructions. The presented results show that a significant overall application speedup is achieved even for large kernels (for ADPCM decoder the speedup ranges from x1.2 to x3.7) and that our algorithm compares well with other state-of-art algorithms for automatic instruction set extensions.

Categories and Subject Descriptors

C.0 [**General**]: Instruction set design; G.1.6 [**Optimization**]: Integer Programming; G.2.2 [**Graph Theory**]: Graph algorithms

General Terms

Algorithms, Performance, Design

Keywords

Instruction-Set Extension, HW/SW Codesign, Reconfigurable Computing

1. INTRODUCTION

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

In this paper we introduce a general and efficient algorithm that selects the new functionalities to be executed in hardware for improving the overall performance. The proposed algorithm targets the Molen organization [14] which allows for a virtually unlimited number of new instructions without limiting the number of input/output values of the function to be executed on the reconfigurable hardware. The elementary building blocks of the approach are clusters of operations known as Multiple Input Single Output (MISOs). Using efficient LP solvers and synthesis results, the largest identified MISOs, called MAXMISO (denoted as MM), are then combined per level and clustered as new application-specific instructions. The result is a cluster of operations with Multiple Inputs and Multiple Outputs, called MIMO, which is executed on the reconfigurable hardware and which provides the maximum performance improvement under reconfigurable hardware resource constraints. The presented results show that a significant performance gain is achieved by hardware execution of the selected new instructions (up to x3.7 for the ADPCM Decoder). More specifically, the main contributions of this paper are:

- construct convex MIMO based on MAXMISOs clustering in order to maximally exploit the MAXMISO level parallelism. Single MAXMISOs usually do not provide significant performance improvement. Thus, we propose MAXMISOs combination in order to take advantage of the parallelism inherent to the hardware execution and the theorem in Section 3.2 that guarantees the MIMO convexity by construction.
- formulation of the instruction selection algorithm as a global ILP problem, where the objective function is the minimization of the execution time and the constraints represent the limited hardware area.
- elimination of the restrictions of the types and number of new instructions (in contrast with most of the existing approaches): there is no limitation on the number of input/output values or the number of new instructions.
- the proposed approach is not restricted to basic-block level analysis but can be applied directly to large kernels.

The paper is organized as follows. In Section 2, we discuss background information and related work. In the following section, we present the theoretical contribution and the ILP problem formulation. Results and the experimental setup are discussed in Section 4 and finally we present conclusion and future work.

2. BACKGROUND AND RELATED WORK

The algorithms for automatic 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 automatic instruction set extensions which are briefly presented in this section.

For the first category, a representative example is introduced in [1] which addresses the generation of MISO instructions of maximal size, called MAXMISO. The proposed algorithm exhaustively enumerates all MAXMISOs. Its complexity is linear with the number of nodes. The reported performance improvement is of few (four) processor cycles per newly added instruction. The approach presented in [9] targets the generation of general MISO instructions. The exponential number of candidate instructions turns into an exponential complexity of the solution in the general case. In consequence, heuristic and additional area constraints are introduced (e.g limitation of only 5-inputs is imposed) to allow an efficient generation. The difference between the complexity of the two approaches is due to the properties of MISOs and MMs: while the enumeration of the first is similar to the subgraph enumeration problem (which is exponential) the intersection of MMs is empty and then once a MM is identified, it is removed generating a linear enumeration of them.

The algorithms included in the second category are more general and provide more significant performance improvement. However they also have exponential complexity. For example, in [4] the identification algorithm detects optimal convex MIMO subgraphs but the computational complexity is exponential. A similar approach described in [15] proposes the enumeration of all the instructions based on the number of inputs, outputs, area and convexity. The selection problem is not addressed. In [3] the authors target the identification of convex clusters of operations given input and output constraints. The clusters are identified with a ILP based methodology similar to ours. The main difference is that they iteratively solve ILP problems for each basic block, while in our approach we have one global ILP problem for the entire procedure. Additionally, the convexity is addressed differently: in [3], the convexity is verified at each iteration, while in our approach it is guaranteed by construction based on the theorem from Section 3.2. Other approaches cluster operations considering the frequency of execution or the occurrence of specific nodes [11, 13] or regularity [6]. Still others impose limitation on the number of operands [5, 2, 8] and use heuristics to generate sets of custom instructions which therefore can not be globally optimal.

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

3. MM-LEVEL ALGORITHM

In this section, we first introduce a motivational example to informally sketch the main concepts of the proposed algorithm. Next, we present the theoretical foundation of the algorithm, followed by a formal problem statement and finally we present in detail the steps of the MM-Level algorithm.

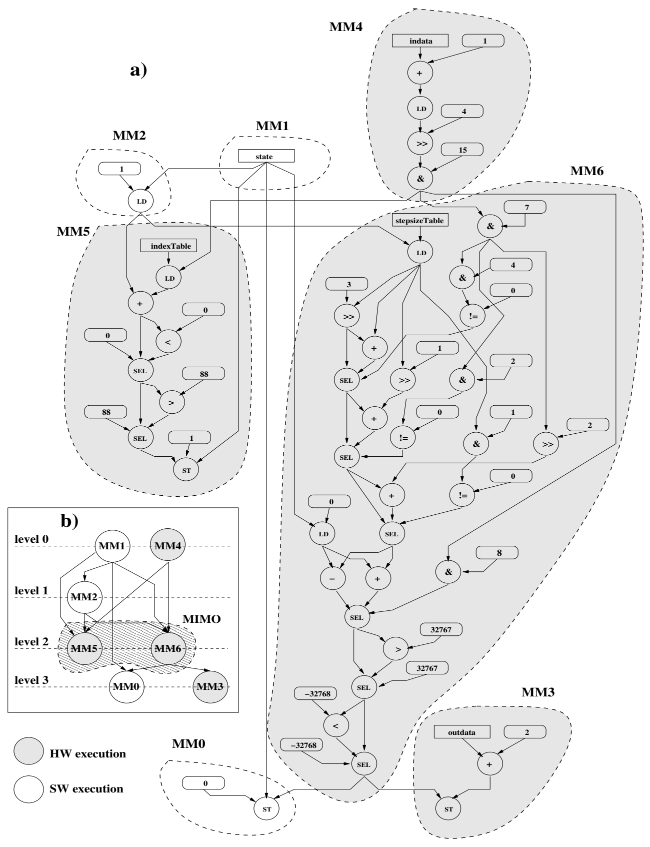

Figure 1: Motivational example: dataflow subgraph from *ADPCM* Decoder with a) MAXMISO identification and b)reduced graph

3.1 Motivational Example

In Figure 1, we present the dataflow subgraph of the ADPCM decoder from the ADPCM application as implemented in the MediaBench benchmark suite [12]. In the first step, our algorithm identifies the MAXMISOs which cover the input DFG (see Figure 1(a)). Each MAXMISO is collapsed as a single node in the reduced graph presented in Figure 1(b). For real applications, the performance improvements provided by hardware execution of a single MAXMISO is not significant.

Loosely stated, the main idea of our algorithm is to combine MAXMISOs available at the same level in the reduced graph, in a convex MIMO that is executed as a single instruction in hardware. For example, let assume the hardware latencies for MM5 and MM6 to be l5 and l6, respectively. By clustering MM5 and MM6 in a new instruction, the latency of this instruction will be max(l_5, l_6). If MM5 and MM6 are implemented as separate instructions, the execution time for that part of the code is equal to $l_5 + l_6$. Thus, the proposed clustering will provide a significant performance gain, that can be roughly estimated as ($l_5 + l_6$) - max(l_5, l_6). For the general case, when n MAXMISOs are clustered, the execution time of the new instruction is max(l_i), where $i = 1, ...n$. The performance gain in this case is sum(l_i) - max(l_i), for $i = 1, ...n$.

The MAXMISO clustering is limited by the size of the reconfigurable hardware. In order to choose the best clustering which provides the maximal performance gain and still satisfying the resource constraints, we formulate the MAXMISO clustering selection as an ILP problem and use an efficient solver for finding the optimal solution.

3.2 Problem Statement and Algorithm

In order to formally express the problem previously presented, we first introduce the necessary definitions and the theoretical foundation of the solution. We assume that the input dataflow graph is a DAG called $G < V, E >$, where the nodes in V represent the primitive operations and the edges in E represent the data dependencies. The nodes can have two inputs at most and their single output can be input to multiple nodes.

MISO Definition. A subgraph $G^* < V^*, E^* > \subset G < V, E >$ is a MISO with root r, $r \in V^*$, if $\forall\ v_i \in V^*$, $\exists$ a path $path(v_i \rightarrow r)$, and $\forall\ path(v_i \rightarrow r)$, $path(v_i \rightarrow r) \subset G^*$.

MAXMISO Definition. 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.

Convex Graph Definition. A subgraph $G^* \subset G$ is **convex** if there exists no path between two nodes of G^* which involves a node of $G \backslash G^*$.

From the above definitions, it can be easily deduced that any MAXMISO is a convex graph. Let $f : G \rightarrow \hat{G}$ be the function such that $MM_i \subset G \mapsto a_i \in \hat{G}$, i.e. f collapses a MM of G in a node of $\hat{G}$. Clearly $\hat{G}$ has as many nodes as the number of MMs in G. The function f is called a **collapsing function**.

Graph Level Definition. For a graph $G < V, E >$, $Level : V \rightarrow \mathbb{N}$ is the function defined as follow:

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

The depth d of a graph $\hat{G}$ is the maximum level of its nodes. The **level** of a MM, $Level(MM_i)$, is defined as the level of $a_i = f(MM_i)$ in $\hat{G}$.

Theorem. Let G be a DAG, $A_1,\ A_2 \subset G$ are two MMs[1] and $Level(A_1), Level(A_2)$ represent the level of A_1 and A_2 respectively. Let $C = A_1 \cup A_2$. If

$$Level(A_1) = Level(A_2) \qquad (1)$$

then C is a convex disconnected MIMO.

Proof. Let G be decomposed as union of MMs, let $f : G \rightarrow \hat{G}$ be the collapsing function. Let a_1, a_2 and c be the images through f of A_1 and A_2 and C respectively. f transforms equation 1 in $Level(a_1) = Level(a_2)$. By contradiction, if $a_1 \cup a_2$ is not convex, then there exists at least one edge from a_1 to a_2 (or from a_2 to a_1). Then $Level(a_2) = Level(a_1) + 1$ (or $Level(a_1) = Level(a_2) + 1$) which contradicts the assumption $Level(a_1) = Level(a_2)$. As a result c is a disconnected graph and then considering the uncollapsing function $f^{-1} : G^* \rightarrow G$, such that $c \mapsto f^{-1}(c) = C$ is a disconnected convex MIMO graph.

[1]Clearly $A_1 \cap A_2 = \emptyset$, [1].

Corollary. Any combination of MMs at the same level is a convex MIMO.

Proof. Assuming that the combination of MAXMISOs at the same level is not a convex MIMO then there exists at least one path between two MAXMISOs that is not included in the final MIMO. This contradicts the previous theorem. Thus, the initial assumption is not true. Hence, the corollary is proven.

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

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

$$\sum_{n_{ij} \in SW} l_{SW_{ij}} + \sum_{i=0}^{d} \max_{n_{ij}\ \in\ HW} l_{HW_{ij}}, \qquad (2)$$

under the following constraint:

$$\sum_{n_{ij} \in HW} \alpha_{ij} \leq A. \qquad (3)$$

Formula (2) represents the minimization of the total execution time: the first term is the execution time of the the MMs that are executed in software and have a sequential execution, while the second term represents the latency of the MMs that are selected for hardware execution in parallel at each level. The constraint expressed by (3) represents the requirement that all new instructions should fit on the total hardware area available.

MM-Level Algorithm: The problem previously presented can be solved as a $0-1$ linear programming problem to produce an optimal solution using an efficient solver.

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

Objective Function. Following the problem statement, formula (2) can be translated in the following objective function

$$\sum_{n_{ij} \in V} l_{SW_{ij}} * \overline{x_{ij}} + \sum_{i=0}^{d} \max_{n_{ij}\ \in\ V} l_{HW_{ij}} * x_{ij}. \qquad (4)$$

$$
\begin{array}{ll}
\text{min:} & lsw_0 * \overline{x_0} + lsw_1 * \overline{x_1} + lsw_2 * \overline{x_2} + lsw_3 * \overline{x_3} + \\
& lsw_4 * \overline{x_4} + lsw_5 * \overline{x_5} + lsw_6 * \overline{x_6} + \\
& xmax_0 + xmax_1 + xmax_2 + xmax_3 \\
\\
\text{C1:} & a_0 * x_0 + a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + \\
& a_4 * x_4 + a_5 * x_5 + a_6 * x_6 \leq A \\
\text{C2:} & xmax_0 \geq lhw_1 * x_1 \\
\text{C3:} & xmax_0 \geq lhw_4 * x_4 \\
\text{C4:} & xmax_1 \geq lhw_2 * x_2 \\
\text{C5:} & xmax_2 \geq lhw_5 * x_5 \\
\text{C6:} & xmax_2 \geq lhw_6 * x_6 \\
\text{C7:} & xmax_3 \geq lhw_3 * x_3 \\
\text{C8:} & xmax_3 \geq lhw_0 * x_0
\end{array}
$$

Figure 2: ILP problem for the motivational example from Figure 1

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

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

$$\sum_{n_{ij} \in V} l_{SW_{ij}} * \overline{x_{ij}} + \sum_{i=0}^{d} xmax_i. \tag{5}$$

with the additional constraints:
$xmax_i \geq l_{HW_{ij}} * x_{ij}$, $\forall i \in \{0, ..., d\}$ with $n_{ij} \in Level_i$.
Linear System of Inequalities. The original constraint given by (3) can be expressed as follow:

$$\sum_{n \in V} \alpha_{ij} * x_{ij} \leq A. \tag{6}$$

Example: Using the motivational example as presented in Figure 1, the ILP problem formulation is presented in Figure 2. There are seven x-variables, associated with the seven MAXMISOs, and four *xmax*-variables, associated with each level in the reduced graph. The solution of this ILP problem for the *lsw* and *lhw* values, computed as explained in Section 4, and the FPGA XC2VP4 is graphically represented in Figure 1 as HW/SW selection.

Using an efficient solver, the optimal solution specifies which are the HW nodes and which the SW ones minimizing objective function 2. In summary, the steps required for the MM-Level algorithms are the following:

- Step 1: MAXMISO identification: using an algorithm similar to the one presented in [1]
- Step 2: Construction of the reduced graph: each MAXMISO is collapsed on one node
- Step 3: HW/SW estimation: evaluate the HW/SW execution latency for each MAXMISO
- Step 4: ILP problem formulation: identify the objective function and set of constraints
- Step 5: ILP problem solving: select the MAXMISOs which are combined into one new instruction

4. EXPERIMENTAL SETUP AND 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 benchmark applications.

The above described algorithm is part of a larger toolchain that aims to support the hardware designer in the design process. The tool chain for the experiments is presented in Figure 3. The input is C code in which the kernel functions are marked with pragmas. The annotated functions are transformed into data-flow graphs (DFGs). The generated graphs are analyzed for the enumeration of the set of MAXMISOs and the construction of the reduced graphs. The software execution time for each MAXMISO is estimated and the VHDL code is generated for each of the selected MAXMISOs. The produced models are further synthesized and the hardware costs for each MAXMISO in terms of occupied area and delay are recorded. The estimated software and hardware implementation costs are used as input of an ILP problem solver. The output of the solver is a list of MAXMISOs for each level that should be implemented in the hardware in order to speedup the kernel execution. The shadowed blocks in the figure denote tools that have been developed. The C-to-DFG converter is implemented within the SUIF2 [2] compiler framework. The MAXMISO processing tools and the VHDL generation tools are stand-alone console applications. For the synthesis, we used Xilinx [3] ISE 7.1.02i. The used ILP problem solver is Frontline's [4] XPRESS Solver.

The tools in the developed toolchain do not require any manual efforts to be adjusted to the target application. In the current version, all steps in the algorithm in Figure 3 are automated except the integration of the tools. The integration of tools is secured through shell scripts/batch files, depending on the used OS - except the ILP solver that is implemented as MS Excel add-in. Nevertheless, as future work we intend to fully automate the tool chain by implementing the necessary VBA (Visual Basic for Application) modules.

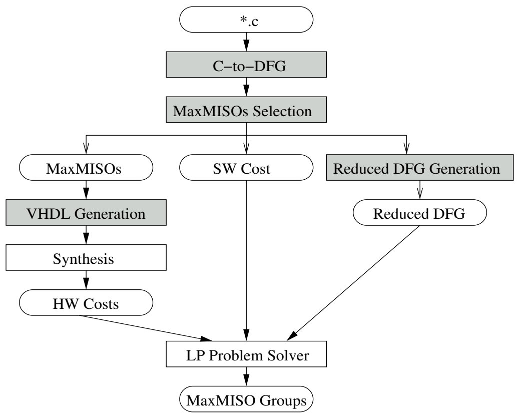

Figure 3: Tool Chain

The software execution time for each MAXMISO is computed as the sum of the latencies of its operations. The hard-

[2]http://suif.stanford.edu/suif/suif2
[3]http://www.xilinx.com
[4]http://www.solver.com

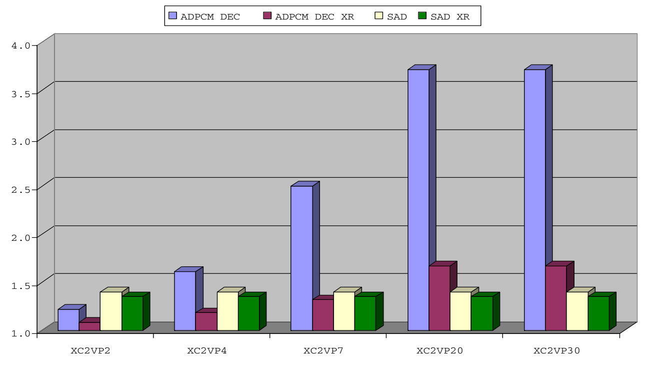

Figure 4: ADPCM Decoder/MPEG2 Encoder overall speedup with the MM-Level algorithm for different FPGA sizes

ware 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. Therefore, the software execution is assumed to be performed on PowerPC 405 operating on 300MHz[5] and the VHDL synthesis is performed for XC2VP30-7 chip[6]. The PowerPC processor in VirtexII Pro does not provide floating-point instructions. Therefore, the floating-point operations in the benchmark suite kernels are converted to the proper integer arithmetic[7]. The DCT, IDCT and ADPCM decoder kernel have been unrolled by a factor of 8/16 in order to increase the selection space of our algorithm. The MOLEN organization and implementation does not impose any restrictions of the operations included in the custom instructions. Hence, the selected operations are not limited only to arithmetic and logic operations, rather they can contain also memory accesses and control logic. Therefore, additional modifications of the source code before the selection process are not necessary.

In the current MOLEN prototype, the access to the Exchange Registers[8] (XRs) for the input/output values is significantly slower compared to the GPP registers. 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 [12] benchmarks and MJPEG encoder application. Table 1 shows the selected applications, the processed kernels and the number of nodes of the original (column 3) and reduced (column 4) graphs. An important observation is that for the considered kernels, algorithms with exponential computational complexity are hard to apply. The listed kernels are selected through profiling using the Virtutech's [9] Simics simulator and the input data included in the benchmarks.

Table 1: Benchmarks and Kernels

Application	Kernel	DFG nodes	R-DFG nodes
MPEG2 Enc	SAD	182	1
MPEG2 Dec	IDCT	2636	578
MJPEG Enc	DCT	6664	388
ADPCM Dec	ADPCMDec	1175	99

Table 2: Execution Time

Tool	Execution Time
MAXMISO extraction[11]	$\sim$ 10 sec
ILP problem generation[11]	$\sim$ 10 sec
ILP problem solver[12]	$<$ 2 min
VHDL generation[11]	$<$ 1 min
RTL synthesis[13]	$<$ 5 min

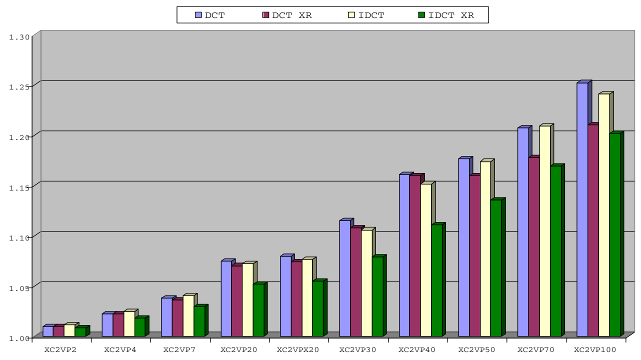

Figure 5: MJPEG Encoder/MPEG2 Decoder overall speedup with the MM-Level algorithm for different FPGA sizes

In Figures 4 and 5, we present the overall application speedup for FPGAs of various sizes compared to the pure software execution.[10] In Figure 4, we only presented a small set of FPGAs as a further increase of the FPGA size does not provide additional improvement. The results in Figure 5 include data for larger set of FPGAs. The speedup estimation is 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 x3.7 for the ADPCM Decoder and different FPGA sizes. For the rest of the considered applications, the speed up is up to x1.6, since the execution time spent in these kernels does not compare to the ADPCM Decoder.

One first and expected observation is that the impact on performance of the MM-Level algorithm is increasing with the size of the available FPGA area. This is explained by the fact that more MAXMISOs can be selected for parallel exe-

[5]We assume a simplified software execution model where caches, pipelines, etc. are not considered.
[6]The generated VHDL is not optimized and hardware reuse is not addressed.
[7]Please, note that this is not limitation of the selection process, rather on the HW and SW implementation of the operations.
[8]Used for GPP-FPGA communication.
[9]http://www.virtutech.se

[10]The occupied area is not shown, since it is almost equal to the available area on the FPGAs.
[11]In-house developed tool.
[12]Frontline's XPRESS Solver, www.solver.com
[13]Xilinx ISE 7.1.02i, www.xilinx.com

cution 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. One important notice is that for the ADPCM benchmark, the speedup provided by our algorithm (x 3.7) is similar to the speedup (x 3.5) achieved for the same kernel and reported in [4] for state-of-the-art algorithm for automatic Instruction Set Extensions. However, we emphasize that our approach does not impose limitations on the number of operands and the number of new instructions.

Regarding the execution time of the presented algorithm, we show some average measurements in Table 2. For most of the considered kernels and FPGAs sizes, the HW selection takes less than 1 minute. However, in a few cases, the best solution found after two minutes spent by the ILP solver is considered.

As a final remark, we mention that the proposed algorithm is particularly profitable for the cases when MAXMISOs have a relatively small number($\sim$15) of nodes. However, in the case when MAXMISOs are large and frequently repeated, the ILP problem formulation can be easily extended to optimally select single MAXMISOs which will be implemented as separate instructions.

5. CONCLUSION

In this paper, we have introduced an algorithm which selects clusters of MAXMISO for execution as a new application-specific instruction on the reconfigurable hardware. One of the main contributions of this paper is the formulation of the instruction selection as an ILP problem for the minimization of the execution time under hardware area constraints. Additionally, the proposed algorithm is general: new instructions have no limitation of their types and numbers and the algorithm is applied beyond basic block level. To the best of our knowledge, this is the first approach that combines MAXMISOs in convex MIMOs. The used SW model in the experiments is simplified not reflecting the available processor optimizations (pipelines, cache hits, etc). However, we do not consider also the possible penalties like branch miss-predictions, cache misses, etc. In addition, the generated VHDL is not optimized. Therefore, we consider that the significant estimated speedup, shown in the results, proves the advantages of our approach. The presented results show significant estimated speedup. In our future work we intend to introduce a more general MM clustering - not limited at a single level in the reduced DFG. Additionally, we will extend the available compiler in order to automatically recognize the selected clusters and generate the appropriate instructions.

6. REFERENCES

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

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

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

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

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

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

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

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

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

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

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

[12] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *International Symposium on Microarchitecture*, pages 330–335, 1997.

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

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

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