A Framework for the Automatic Generation of Instruction-Set Extensions for Reconfigurable Architectures

Carlo Galuzzi and Koen Bertels*

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

Abstract. In this paper we present a framework for the automatic identification and selection of convex MIMO instruction-set extensions for reconfigurable architecture. The framework partitions the analysis of the problem into phases of different computational complexity and it generates instruction-set extensions of different granularity. The framework is retargetable and additional clustering policies can be added with just small modification on the design.

1 Introduction

In the past decade we have witnessed a general shifting from the use of general-purpose computing systems to systems able to perform only a limited number of tasks but more efficiently. Although general-purpose systems 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 represented by reconfigurable systems. A simple reconfigurable system 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 system, a certain number of instructions are executed in hardware, namely the ones that belongs to the Instruction-Set, whereas the rest of the instructions is executed in software. If the same application is executed on a reconfigurable system, we can use the reconfigurable hardware to execute additional more complex instructions, application-dependent, so that 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* [2].

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

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

R. Woods et al. (Eds.): ARC 2008, LNCS 4943, pp. 280–286, 2008.

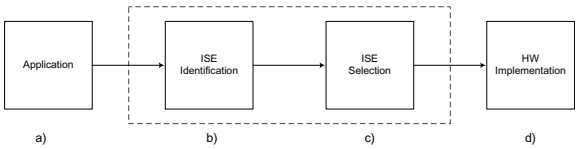

Fig. 1. The main parts of the ISE creation process: a) application to analyze, b) extension identification c) extension selection and d) hardware implementation of the selected new instructions

implementation[1], the problem becomes the design of efficient algorithms for the identification of only instructions suitable for a hardware implementation.

Figure 1 depicts a general flow for ISE identification: once the application is selected (Figure 1a), the application is analyzed to discover a certain number of candidate instructions for hardware implementation (Figure 1b), the identified instructions pass through a selection process which identify the most suitable ones to hardwire usually based on hardware limitations (Figure 1c) and finally the selected instructions are implemented in hardware (Figure 1d).

In this context, we present *a framework for the automatic identification and selection of Multiple Input Multiple Output Instruction-Set extensions*. The proposed design 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 new instruction to be executed on the reconfigurable hardware. More specifically the main contributions of this paper are the below listed:

- a framework for the automatic identification and selection of Instruction-Set extensions which partitions the analysis into phases of different granularity and computational complexity;
- an analysis of the main issues to export the presented framework, designed for the Molen architecture, to general reconfigurable architectures.

The reminder of the paper is the following: in Section 2, a description of the framework in detail together with a computational complexity analysis and an analysis of the issues to improve the presented framework and extend it to a general reconfigurable architectures are presented. Finally, in Section 3, we present concluding remarks and an outline of future work.

2 Description of the Framework

In Figure 2, we present an overview of the framework proposed in this paper. For more technical details about definitions and concepts addressed in the following,

[1] Depending on the target architecture, the new instructions can have limitations on the total number of inputs and/or outputs, or on the area they occupy when implemented on the reconfigurable hardware, etc.

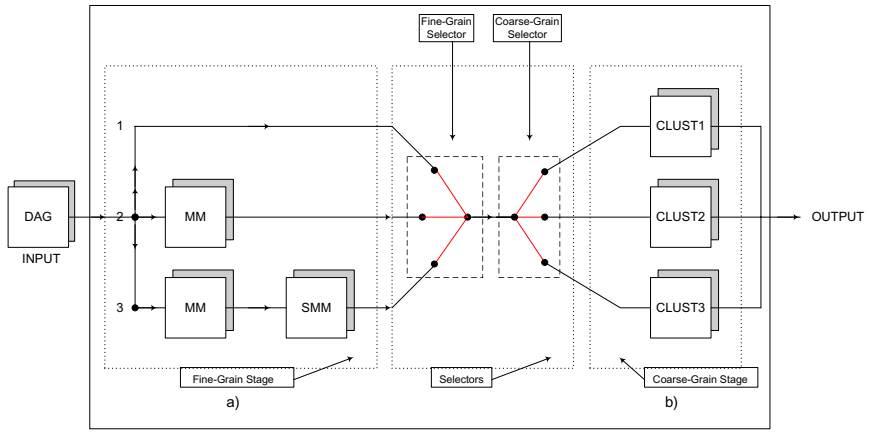

Fig. 2. The Framework for the Automatic Generation of Instruction-Set extensions

the interested reader can refer to [5,6,4,7]. The main idea behind the design of such a framework is the generation of convex MIMO instruction-set extensions with multiple steps of different granularity and complexity. More specifically the design can be split in two parts: the first part concerns a fine-grain clustering (Figure 2a) and the second one concerns a coarse-grain clustering (Figure 2b). The framework has three main stages: the fine-grain stage, the selector stage and the coarse-grain stage described in the following. We assume the Molen architecture as the target reconfigurable architecture. Some issues concerning how to export the framework to general reconfigurable architecture and possible extensions and improvements for the framework are addressed later in the paper.

One of the strengths of such a framework is the convexity guarantee of the clusters generated. Traditional methods for convex instruction identification usually perform a check for the convexity of the cluster at each inclusion of a node in the cluster. This affects the overall complexity of the solution which increases. With our clusterings, the selected nodes do not have to be tested for convexity since the convexity of the cluster is theoretically guaranteed by construction.

STEP 1: Fine-Grain Stage. The input of the framework is given by the DAG $G = (V, E)$ that represents the application to be analyzed where the nodes represent primitive operations and the edges represent the data dependencies. The focus of the analysis is on which parts of the code are suitable for a software implementation and which ones are suitable for a hardware implementation to increase the overall performance of execution of the application onto the reconfigurable architecture. We have three different paths that the input can follow before arriving to the selector stage:

- PATH 1: The nodes of G are convoyed directly to the selector step;
- PATH 2: The nodes of G are partitioned in MAXMISOs [3];
- PATH 3: The nodes of G are firstly partitioned in MAXMISOs and subsequently every MAXMISO is partitioned in SMMs [4].

In PATH 3, from the definition of SMM [4], we know that the SMM partitioning depends on the choice of the bereaved node[2] and this node can be selected in n different ways, where n is the order of G. In all the cases the output is a minimal cover of convex non-overlapping MISO elements[3]. We remind that a *minimal cover* is a cover for which removal of one member destroys the covering property of the graph. *We note that in all cases convexity is guaranteed theoretically by construction* while in general, other approaches perform an additional analysis on the clusters to test the convexity. We remark that, in PATH 3, the algorithm for SMM generation can also be used iteratively to generate MISO instructions of relatively smaller size when tight constraints on the inputs are imposed.

STEP 2: Coarse-Grain Stage. Based on the output of the fine-grain stage (more specifically on the input selected by the fine-grain selector) there are three possible ways to generate convex MIMO instructions depending on the shape of the graph in terms of depth and width of the graph[4].

CASE 1: $w > d$. (CLUST 1, Figure 2) Let us consider a partitioning of the nodes of G in levels. When two nodes n_1 and n_2 with latency in hardware l_1 an l_2 at the same level are selected for hardware execution, if they are implemented as separate instructions we have a performance loss, which can be roughly estimated as $l_1 + l_2 - \max(l_1, l_2)$. An optimal selection of which nodes to select at the same level to implement in parallel in hardware can then provide a considerable speed up estimated as $\sum_i l_i - \max_i(l_i)$. An algorithm for the optimal selection of convex MIMO ISE based on an ILP formulation, more suitable for graphs wider than deeper, has been proposed in [5]. Although the algorithm has been designed for an optimal selection through the levels of MAXMISOs at the same level, it is possible to generalize the result to every minimal cover with non-overlapping elements.

CASE 2: $w < d$. (CLUST 2, Figure 2) When a graph is deeper than wider, a heuristic clustering algorithm of linear complexity in the number of processed elements is proposed in [6]. Similarly to the previous case, the result is applicable to every minimal cover with non-overlapping elements. This algorithm starts from a node at a certain level and moving vertically through the levels it identifies nodes to include in the cluster. Clustering is performed up to when there is available area left on the reconfigurable hardware.

CASE 3: $w \propto d$. (CLUST 3, Figure 2) When clustering is performed on graph with comparable width and depth, an extended version of the algorithms proposed in the previous cases is presented in [7]. This paper present a clustering method of linear complexity based on a spiral search through the levels of a

[2] The choice of the node can be random or directed by specific properties defined by the user.

[3] We note that a single node is trivially a convex subgraph and a MISO.

[4] The depth, d, of a graph is defined as the maximum number of the levels of its node, while the width, w, of a graph is defined as the maximum number of nodes belonging to the same level through the levels.

graph. Contrary to the previous two algorithms which select nodes favoring a specific direction (horizontal CASE 1 and vertical CASE 2), this algorithm clusters nodes following a spiral search centered in the initial node selected and expanding the search through the levels in both directions: vertical and horizontal. Also in this case clustering is performed up to when there is available area left on the reconfigurable hardware. Similarly to the previous cases, the result is extendible to every minimal cover with non-overlapping elements.

These algorithms perform instruction generation and selection at the same time based on a certain number of parameters: hardware and software latency of the generated instructions, total area occupied by the generated instructions when implemented in hardware and total area available on the FPGA. Additionally, while the first clustering produces an optimal solution, the other clusterings are heuristics. We remark that although the algorithms are more suitable in specific cases than others, there is no limitation in the use of any of them for any graph.

Fine- and Coarse-Grain Selector Stage. In Figure 2 two selectors are depicted: a fine-grain and a coarse-grain selector. The former, a 3-1 selector, forwards the output of one of the PATH 1-3 to the latter, a 1-3 selector, which directs the data to one of the coarse-grain clustering algorithms. We have 3×3 possible combinations, which means that we can have up to 9 possible different instruction-set extensions of different granularity. Additionally the framework can be extended with additional algorithms for clustering in both stages, the fine-grain and the coarse-grain stage, with small adjustments on only the selectors to include more inputs or outputs for the additional clustering algorithms.

The Complexity of the Framework. As described before, the framework produces an ISE depending on the input and output of the selectors. All the clustering algorithms presented in the framework but one have linear complexity in the number of processed elements. Only the clustering algorithm CLUST 1 has exponential complexity but it provides an optimal solution. This means that when in the coarse-grain stage it is selected the first algorithm for the generation of convex MIMO instruction set extensions, the overall complexity of the process is exponential. In the remaining cases the overall complexity is linear in the number of processed elements.

Additionally, the SMM clustering, CLUST 2 and 3 generate clustering based on the initial selection of a node, which can be random or directed by specific properties defined by the user. This means that keeping variable the selection of the node, for each choice of the node it is possible to generate a different instruction-set extension with the same computational complexity.

Extensions and improvements for the Framework. The framework presented in this paper has a flexible design: additional clustering algorithm can be integrated into the design with modifications of only the selectors in principle. As mentioned before, our target architecture is the Molen architecture. When the present design is exported on different architectures, additional constraints on number of I/O have to be usually introduced during the clustering in the fine- and coarse-grain clustering step.

Hardware reuse can be considered to further speed up the overall execution time of the application onto the reconfigurable architecture, implementing in hardware only the unique instructions and saving area for additional ones. This can be done with an isomorphism check strategically positioned into the design. On one side, an isomorphism check for hardware reuse can save area and increase the speed up using the saved area for additional new complex operations. On the other side, no polynomial solution is known for the graph isomorphism problem. This means that the inclusion into the design of an isomorphism check will increase the overall complexity of the solution. Efficient algorithms for graph isomorphism are available in literature, which represents a good trade-off between their complexity and the quality of their solution [8].

Additionally, CLUST 2 and CLUST 3 perform selection of the clusters based on the total available area left. An optimal selection of which are the instructions suitable for hardware implementation based on latency and area can be obtained formulating the selection as an ILP problem and using efficient solver to find the solution. This can be solved as in [5] without the requirements that the selected clusters belong to specific levels. This will provide a better selection of the instruction but it will increase the overall complexity of the generation process as well. This means that all the clusters will be first generated, giving a minimal cover of the graph and then the ones belonging to the optimal solution will be implemented in hardware based on the total available area on the reconfigurable component.

3 Conclusions

In this paper we presented a framework for the automatic identification and selection of convex MIMO instruction-set extensions for reconfigurable architecture. The framework partitions the analysis of the problem into phases of different computational complexity and granularity. The framework is retargetable and additional clustering policies can be added with just small modification on the design. In our future work we intend to verify with experimental results the benefit of the insertion of such a framework into the design for automatic instruction set extension. Preliminary results presented in [5] and [6] have shown the benefit of the insertion of part of the algorithms presented in this paper.

References

1. Vassiliadis: The molen polymorphic processor. IEEE Trans. on Comp. 53(11) (2004)
2. Galuzzi: The Instruction-Set Extension Problem: A Survey. In: ARC 2008 (2008)
3. Alippi: A dag-based design approach for reconfigurable vliw processors. In: DATE 1999 (1999)
4. Galuzzi: A linear complexity algorithm for the generation of multiple input single output instructions of variable size. In: SAMOS VII

5. Galuzzi: Automatic selection of application-specific instruction-set extensions. In: CODES+ISSS 2006 (2006)
6. Galuzzi: A linear complexity algorithm for the automatic generation of convex multiple input multiple output instructions. In: Inter. J. of Elec. (2008) (to appear)
7. Galuzzi: The spiral search: A linear complexity algorithm for the generation of convex multiple input multiple output instruction-set extensions. In: ICFPT 2007 (2007)
8. Bonzini: A retargetable framework for automated discovery of custom instructions. In: ASAP 2007 (2007)