Compiler-driven FPGA-area Allocation for Reconfigurable Computing

Elena Moscu Panainte Koen Bertels Stamatis Vassiliadis

Computer Engineering
Delft University of Technology, The Netherlands
http://ce.et.tudelft.nl
E-mail: {E.Panainte, K.Bertels, S.Vassiliadis}@ewi.tudelft.nl

Abstract

In this paper, we propose two FPGA-area allocation algorithms based on profiling results for reducing the impact on performance of dynamic reconfiguration overheads. The problem of FPGA-area allocation is presented as a 0-1 integer linear programming problem and efficient solvers are incorporated for finding the optimal solutions. Additionally, we discuss the FPGA-area allocation problem in two scenarios. In the first scenario, all hardware operations are allocated on the FPGA while in the second scenario, any hardware operation can be switched to software execution in order to provide an overall performance improvement. We evaluate our proposed algorithms using the MPEG2 and MJPEG encoder multimedia benchmarks and the hardware implementations for SAD, DCT, IDCT, Quantization and VLC tasks. We show that a significant performance improvement (up to 61 % for MPEG2 and 94 % for MJPEG) is to be achieved when the proposed algorithms are used, while the reconfiguration overhead is reduced by at least 36 % for MJPEG.

1. Introduction

Although the new generations of FPGAs provide support for partial and dynamic configuration, the huge reconfiguration latency is still a major shortcoming of the current FCCMs (see [7]). In this paper, we propose two FPGA-area allocation algorithms for the tasks executed on the reconfigurable hardware. The goal is to minimize the FPGA-area which is reconfigured at runtime and improve the overall performance, taking into account the application runtime features. More specifically, we use the reconfiguration frequency for the target application to guide the allocation algorithms. Two scenarios are discussed: the first one corresponds to the case when all hardware operations must be placed/executed on the target FPGA while in the second scenario, a hardware operation can be switched to its pure software execution on the core processor in order to reduce the pressure/competition for the FPGA area. The FPGA-area allocation problem is formulated as a 0-1 integer linear programming (LP) problem and efficient LP solvers are used for finding the optimal solutions.

The paper is organized in six sections. The background and related work is presented in the following section. Next, we discuss some motivational examples and define the FPGA-area allocation problem addressed in this paper. The proposed allocation algorithms are detailed in section 4. Finally, we provide the evaluation of the proposed algorithms and present conclusions and future work.

2. Background and Related Work

In this paper, we assume the Molen programming paradigm [9] for FCCMs (Field-programmable Custom Computing Machines) with a core processor (GPP) and reconfigurable hardware (usually implemented as an FPGA). The reconfigurable hardware is controlled by two instructions: i) SET for the FPGA configuration for a reconfigurable operation (Rop) and ii) EXECUTE for the Rop execution on the FPGA. The Molen compiler [7] generates code for reconfigurable computing platforms following the Molen programming paradigm. An important compiler optimization (see [7]) included in the Molen compiler is to reduce the redundant SET instructions taking into account the predefined FPGA-area conflicts between the considered Rops. In consequence, the compiler optimization will benefit from an efficient FPGA-area allocation that minimizes the FPGA-area overlaps for a target application.

Previous approaches for FPGA-area allocation are mainly focused on cases where the whole application is decomposed in tasks which are all executed on the FPGA. In [3], an optimal module placement based on packing classes is proposed. A backtracking solution with bounding heuristics is presented in [5]. The proposed solutions require detailed information (such as data flow

3-9810801-0-6/DATE06 © 2006 EDAA

graphs, dependency graphs of tasks) about the application's features and regular application behavior. Another approach (see [4] [10] [2]) is the task allocation in an operating system for reconfigurable computing. In such cases, information about specific application behavior cannot be used in order to guide this allocation, thus optimization opportunities can be lost. Other related work [7] addresses compiler optimization for reducing the number of redundant FPGA configurations based on a predefined FPGA-area allocation. In the current paper, we propose two FPGA-area allocation algorithms that reduce furthermore the number of FPGA configurations by minimizing the total reconfigured area for a given trace of execution.

3. Problem Overview and Definition

Motivational Example: In order to clearly define the FPGA-area allocation problem, we use a motivational example (Figure 1(a)) which sketches an FPGA device and the area requirements for three operations implemented on the FPGA. In this paper, we assume FPGAs with column-based reconfiguration (the reconfiguration may only be performed for a full column of CLBs of the chip) such as the well-known Xilinx Virtex devices. For one application that uses the three hardware operations, a simple FPGA area allocation (presented in 1(b)) places all operations starting with the first column. Due to the FPGA area overlaps, such allocation requires the FPGA reconfiguration before each execution of the considered operations. As shown in [7], FPGA reconfiguration is slow and thus, repetitive FPGA reconfigurations can produce a significant performance decrease. In consequence, a better FPGA-area allocation is required in order to reduce the reconfiguration overhead. An allocation strategy is possible only when the placement of the hardware operations is not predefined.

Two important observations can be made regarding the example from Figure 1. The first observation is that the three considered operations cannot fit together on the FPGA as the sum of the area of their hardware implementations exceeds the total available FPGA-area. The second observation concerns the simple allocation strategy, where there is unused FPGA-area while parts of the FPGA have to be reconfigured before each execution. For the considered example, even when the Rop2 and Rop3 do not have overlapping FPGA-area, the placement of Rop1 will introduce FPGA-area overlaps with one of the two operations.

In order to determine an efficient FPGA-area allocation, we propose an approach that divides the hardware operations in two categories: FIX and RW. An operation is called FIX if it has no overlapping area with any other hardware operations in the considered application. Such a FIX operation requires only one initial FPGA configuration (which

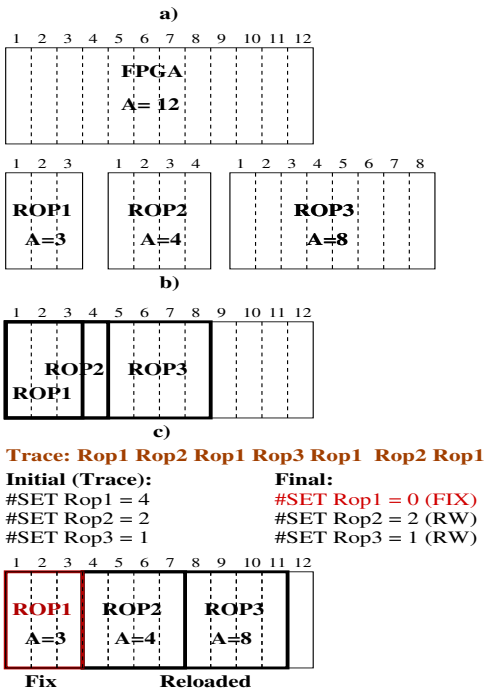

Figure 1. **Example: a) Total FPGA-area and three Rops; b) a simple FPGA-area allocation c) optimal allocation based on the execution trace**

can be preloaded and can be neglected). An operation is called RW (reconfigurable) if its area overlaps with other operations and it has to be configured before each execution. Loosely stated, the main idea of our approach is to minimize the reconfigured FPGA-area based on the reconfiguration frequency of each operation. Using profiling information, we determine the execution order for the hardware operations (called trace) and compute the reconfiguration frequency in the trace. The goal is to allocate the larger and frequently reconfigured operations as FIX operations. The example shown in Figure 1(c) presents the optimal FPGA-area allocation for a given execution trace. We can observe the elimination of hardware configurations for the operations allocated as FIX operations (Rop1 in this example). The selection of the FIX operations is based on 0-1 linear programming and is explained in Section 4. The used terminology and a formal description of the allocation problem is presented in the rest of this section.

Problem statement: We represent a set of n reconfigurable hardware operations (Rops) as $ROP = \{Rop_1, Rop_2, ..., Rop_i, ..., Rop_n\}$, where each operation Rop_i occupies for its hardware implementation an FPGA-area A_i. The total available area of the target FPGA device is S. Although in this paper we address the case when the reconfiguration is column-based

(the area is expressed as the number of columns), the extension to the 2D or 3D cases is straightforward. An execution trace is a sequence of Rops that are executed for a set of representative input data for the target application and it is represented as $T : Rop_i, Rop_j, ..., Rop_k, ...$ A trace is normalized if it does not contain two identical consecutive Rops. This normalization represents the fact that consecutive hardware reconfigurations for the same Rop are redundant and can be eliminated by compiler optimization (see [7]) or hardware prefetching. For each $Rop_i \in ROP$ and a normalized trace T, the reconfiguration frequency $n(T)_i$ represents the number of occurrences of Rop_i in the trace T.

As previously explained, the idea of our approach is to divide the ROP set in two subsets FIX and RW, where $ROP = FIX \bigcup RW$ and $FIX \bigcap RW = \emptyset$. The Rops in the FIX set will have a dedicated area allocated on the FPGA that is not used by other Rops (they do not have area overlaps with other Rops). The advantage is that the FIX Rops will not require an FPGA reconfiguration before their executions. The total area occupied by the FIX Rops is $\sum_{Rop_j \in FIX} A_j$. The Rops in RW set are the operations that have area overlaps. The reconfiguration overhead is proportional with the FPGA-area which is reconfigured at runtime. The aim is to minimize the total reconfigured area (the sum of the area of the Rops from RW multiplied by their reconfiguration frequency) which corresponds to the minimization of the reconfiguration overhead and implicitly, to the improvement of the overall performance gain. A formal description of this problem is as follows:

Problem Given a set $ROP = \{Rop_1, Rop_2, ..., Rop_i, ..., Rop_n\}$, a total available FPGA-area S, a normalized execution trace T, each Rop_i having an FPGA-area A_i and the reconfiguration frequency $n(T)_i$, find $RW \subseteq ROP$ that minimizes the reconfigured area $\sum_{Rop_i \in RW} n(T)_i * A_i$, under the following constraint:

- $\forall\, Rop_k \in RW,\ A_k + \sum_{Rop_j \in FIX} A_j \leq S$, where $FIX = ROP - RW$.

The constraint represents the requirement that any RW Rop must have enough available area to coexists on the FPGA at the execution time with all FIX Rops. Implicitly, as the FPGA-area is a positive number, the constrain expresses also the requirement that all FIX Rops should fit together on the target FPGA. Once the RW set has been determined for the above mentioned problem, an effective FPGA-area allocation is straightforward. Assuming that A_i represents the number of required columns, an FPGA-area allocation associates with each Rops, the number of the first column where A_i is placed. In the first step, the FIX Rops are consecutively allocated on the FPGA. In the second step, the RW Rops are all allocated at the end of the FPGA-area allocated for the FIX Rops.

4. FPGA-area Allocation Algorithms

For the problem defined in the previous section, we propose its formulation as an integer linear pseudo-Boolean (0-1) programming problem and consequently, the solutions can be determined using efficient solvers (see [1]). More specifically, we propose two scenarios. The first case (associated with the FIX/RW Algorithm) corresponds to the above mentioned problem, where the Rops are placed in the FIX or in the RW (Reloaded) part on the FPGA. In the second case (corresponding to the FIX/RW/SW Algorithm), we assume than an Rop can have three options for execution: on the FIX or RW part or additionally, it can be switched to its software execution (on GPP). The last options can be preferred for those Rops where the huge reconfigurations latency consumes the gain produced by the fast execution on the FPGA. In the rest of this section, we introduce in detail the two FPGA-area allocation algorithms.

4.1. FIX/RW Algorithm

As previously presented, we translate the FPGA-area allocation problem in a 0-1 linear programming problem to produce an optimal solution using efficient solvers.

0-1 Selection In the considered case, any Rop can be executed on the FIX or RW part of the FPGA. In consequence, we associate with any Rop_i a variable x_i such that $x_i = \begin{cases} 0 & \text{if } Rop_i \in FIX \\ 1 & \text{if } Rop_i \in RW \end{cases}$. Finding the optimal partition of ROP in FIX and RW is reduced to finding the optimal 0-1 values for all x_i.

Objective function In the problem definition in Section 3, the minimization of the reconfigured area $\sum_{Rop_i \in RW} n(T)_i * A_i$ can be expressed as the following objective function $\sum_{Rop_i \in ROP} n(T)_i * A_i * x_i$. If Rop_i is a FIX Rop, then $x_i = 0$ and it does not increase the reconfigured area as it does not need any configuration. In consequence, only the contribution of the RW Rops is included in the minimization objective function.

Linear Pseudo-Boolean Inequalities The system of linear pseudo-Boolean inequalities of the linear programming problem formulation corresponds to the constrains included the initial problem. The constraint that $\forall\, Rop_k \in RW,\ A_k + \sum_{Rop_j \in FIX} A_j \leq S$ can be expressed as follows:

min:	+2*39*x1	+ 3*13*x2	+ 3*16*x3;	
C1:		+ 13*$\bar{x}$2	+ 16*$\bar{x}$3	$\leq$ 58 - 39
C2:	+ 39*$\bar{x}$1		+ 16*$\bar{x}$3	$\leq$ 58 - 13
C3:	+ 39*$\bar{x}$1	+ 13*$\bar{x}$2		$\leq$ 58 - 16

Figure 2. LP problem for the MPEG2 example in Section 5 and FIX/RW Algorithm

$$\begin{cases} A_1 * x_1 + \sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S \\ A_2 * x_2 + \sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ A_i * x_i + \sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ A_n * x_n + \sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S \end{cases}$$

This system of inequalities should be interpreted as follows: (1) The term $\sum_{Rop_j \in ROP} A_j * \bar{x}_j$ represents the permanently configured FPGA-area occupied by FIX Rops: $\sum_{Rop_j \in ROP} A_j * \bar{x}_j = \sum_{Rop_j \in FIX} A_j * \bar{x}_j$.

(2)The second observation regards the first term in the inequalities, namely $A_i * x_i$. For the cases when $Rop_i \in FIX \Longrightarrow x_i = 0$, the term $A_i * x_i$ can be eliminated. The ith inequality is transformed in $\sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S$ which represents the constraint that the total area allocated for FIX Rops should be smaller or equal than the total available FPGA-area S. Similarly, for the cases when $Rop_i \in RW \Longrightarrow x_i = 1$, the inequality is transformed in $A_i * x_i + \sum_{Rop_j \in ROP} A_j * \bar{x}_j \leq S$ which represents the constraint that an RW Rop has to fit on the FPGA together with all FIX Rops.

In our model implementation, each ith inequality should not contain both x_i and $\bar{x}_i$; thus it can be reduced as follows:

$$A_i * x_i + \sum_{j=1}^{n} A_j * \bar{x}_j \leq S \iff A_i * x_i + A_i * \bar{x}_i + \sum_{j=1}^{i-1} A_j * \bar{x}_j + \sum_{j=i+1}^{n} A_j * \bar{x}_j \leq S \iff \sum_{j=1}^{i-1} A_j * \bar{x}_j + \sum_{j=i+1}^{n} A_j * \bar{x}_j \leq S - A_i$$

Example A real example is presented in Figure 2, for three Rops with $A_1 = 39, A_2 = 13, A_3 = 16, n(T)_1 = 2, n(T)_2 = 3, n(T)_3 = 3$ and $S = 58$. The solution to this problem is $\{x_1 = 0, x_2 = 1; x_3 = 1\}$, which corresponds to $FIX = \{Rop_1\}$ and $RW = \{Rop2, \quad Rop3\}$.

4.2. FIX/RW/SW Algorithm

The FIX/RW algorithm previously presented has two important limitations: i) it cannot find a viable FPGA allocation if there is an Rop_i with $A_i > S$ because the constraint set is unsatisfiable; and ii) although the FPGA execution is (usually) faster than the software execution for any Rop, the reconfiguration overhead can significantly increase the overall execution time. In order to eliminate these rigid limitations, we propose the FIX/RW/SW algorithm where the Rops can additionally be switched to software execution. The FPGA-area allocation problem can again be formulated as 0-1 LP problem including the following components.

0-1 Selection In this case, a Rop has three options for execution: on the FIX or RW part on the FPGA or additionally in software (SW). The allocation problem involves the division of ROP in three subsets FIX, RW and SW, such that $ROP = FIX \bigcup RW \bigcup SW$ and $FIX \bigcap RW = \emptyset$, $FIX \bigcap SW = \emptyset$, $RW \bigcap SW = \emptyset$. These options can be expressed using three boolean variables for each Rop_i, namely $xfix_i, xrw_i$ and xsw_i, where $xfix_i = \begin{cases} 1 & \text{if } Rop_i \in FIX \\ 0 & \text{if } Rop_i \notin FIX \end{cases}$ and similar for xrw_i and xsw_i. Moreover, a Rop must be included in only one subset; this constraint can be expressed as $xfix_i + xrw_i + xsw_i = 1$. Finding the optimal partition of ROP in FIX, RW and SW is reduced to finding the optimal 0-1 values for all $xfix_i, xrw_i$ and xsw_i.

Objective function In the problem definition of the previous FIX/RW Algorithm, the goal of the objective function is the minimization of the total reconfigured area. This function cannot be used in the current scenario as all Rops can be switched to their software execution; thus in the FIX/RW/SW algorithm, the goal is the performance gain. The new objective function is the minimization of the execution time for the considered Rops and is expressed as $\sum_{i=1}^{n} cost\_fix_i * xfix_i + \sum_{i=1}^{n} cost\_rw_i * xrw_i + \sum_{i=1}^{n} cost\_sw_i * xsw_i$, where $cost\_fix_i / cost\_rw_i / cost\_sw_i$ represent the total execution time for Rop_i in FIX/RW/SW respectively and their values can be determined using profiling information and estimations.

Linear Pseudo-Boolean Inequalities The system of linear pseudo-Boolean inequalities of the linear programming problem formulation is similar to the previous FIX/RW system:

$$\begin{cases} A_1 * xrw_1 + \sum_{j=1}^{n} A_j * xfix_j \leq S \\ A_2 * xrw_2 + \sum_{j=1}^{n} A_j * xfix_j \leq S \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ A_i * xrw_i + \sum_{j=1}^{n} A_j * xfix_j \leq S \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ A_n * xrw_n + \sum_{j=1}^{n} A_j * xfix_j \leq S \end{cases}$$

The main idea is the same as in the previous algorithm: each RW Rop must have allocated enough FPGA-area to fit with all FIX Rops on the FPGA.

As a final observation for both algorithms, we notice that the generated FPGA-area allocations will preserve the application semantics even if the input execution trace T is

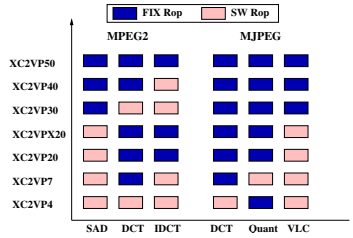

Figure 3. **FPGA allocation for the FIX/RW/SW algorithm**

not a representative trace. In such cases, some performance gain may be lost, but the application has the correct behavior.

5. Results

In this section, we present the evaluation of the performance achieved by the proposed algorithms in the MPEG2 and MJPEG case study.

Target Applications, Rops and FPGA The target C applications considered in this section are the well-known multimedia benchmarks MPEG2 and MJPEG encoders; the input sequence for the MPEG2 is the set of three frames that comes with the benchmark, while for MJPEG we use 30 color frames from "tennis" in YUV format with a resolution of 256x256 pixels. The Rops candidate for execution on the FPGA are i) for MPEG2 - SAD (sum of absolute-difference), 2D DCT (2 dimensional discrete cosine transform) and IDCT (2D inverse DCT) with the real FPGA implementations presented in [9] and ii) for MJPEG - DCT, Quantization and VLC (Variable Length Coding) with the the real FPGA implementations for Quantization and VLC presented in [8]. The target reconfigurable platforms are Xilinx Virtex II Pro devices (see [12]) with CLB array sizes varying from 40 x 22 for XC2VP4 up to 88 x 70 for XC2VP50 and also including one PowerPC processor. The required FPGA-area (expressed in slices) and FPGA execution time (converted in PowerPC at 300 MHz cycles) for the considered Rops are presented in Table 1, columns 2-3. We estimate the FPGA reconfiguration time per CLB based on the total configuration time: 47.55 ms for the whole XC2VP50 chip (CLB array of 88x70) using SelectMAP at 50MHz (as presented in [11]); thus, the reconfiguration overhead (converted in PowerPC cycles) is 2315 cycles per CLB. The basic configuration time for the considered Rops is presented in Table 1, columns 4. For the software execution, the profiling results for computing *cost_sw* for each Rop are based on simulations using the PowerPC simulator from Simics [6]. The time spent for the software execution for the considered Rops reported to the total software execution time is presented in Table 1, last column.

Rop	Slices	EXEC[cycles]	SET[Kcycles]	SW [%]
MPEG2				
SAD	13613	49	7880	62 %
DCT	4314	306	2498	15%
IDCT	5436	315	3146	1 %
MJPEG				
DCT	4314	306	2498	80%
Quant	1179	104	683	3%
VLC	6422	110	3718	12.5 %

Table 1. **HW/SW features for the Rops that candidate for FPGA execution**

FPGA-area Allocation Algorithms Evaluation A comparison between the estimated performance for the MPEG2 / MJPEG encoder applications and the two FPGA-area allocation algorithms is presented in Figure 4. The reference unit of this comparison (SW) is the pure software execution when all Rops are executed on the GPP. We also include in this comparison the performance estimated for the naive FPGA-area allocation presented in Section 3 and denoted as NAlloc for MPEG2. The performance for the proposed algorithms are represented as FIX/RW Alg and FIX/RW/SW Alg. The corresponding solutions for the FIX/RW/SW algorithm are graphically represented in Figure 3. In all cases, we considered that only one FPGA reconfiguration is performed before a sequence of consecutive Rop executions. Otherwise, in the case when an FPGA configuration is performed before each Rop execution, the overall performance is decreased by several orders of magnitude (see [7]). For both algorithms, we use an efficient LP solver implementation based on Davis-Putman enumeration methods presented in [1] and publically available as a software package.

From Figure 4, we notice that the FIX/RW algorithm does not generate solutions for the FPGAs with relatively small CLB arrays (as explained in Section 4.2), while FIX/RW/SW algorithm guarantees that a better (or equal, in the worst case) solution compared to SW is selected. However, for the FPGA devices with large CLB arrays both algorithms select the best solution - all Rops allocated as FIX Rops - which corresponds to an overall performance improvement of 61 % for MPEG2 and 94 % for MJPEG. In an example scenario using the FIX/RW algorithm for the MPEG2 application and XC2VP40 device where the partial and dynamic hardware configuration is needed, it can be observed that the reconfiguration overhead is reduced by 47 %. For the MJPEG application, the reconfiguration overhead is reduced in all cases by at least 36 %. In Figure 3, we notice, that FIX/RW/SW algorithm does not select RW Rops, but SW or FIX Rops are preferred. This observation is explained by the huge reconfiguration latency of the considered devices. Addition-

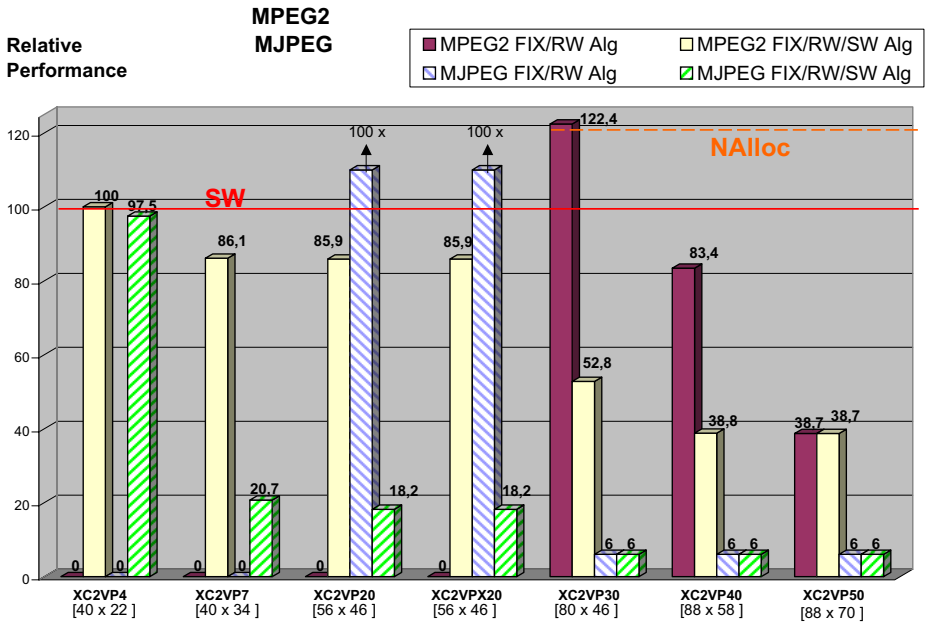

Figure 4. **Performance comparison for the FPGA-area allocation algorithms**

ally, we determine that for the MPEG2 application and XC2VP30 device the RW Rops are used only when the reconfiguration latency is at least 10 times smaller than the current values. In consequence, the FPGA reconfiguration must be at least one order of magnitude faster for an efficient dynamic FPGA usage.

6. Conclusions

In this paper, we have presented two FPGA-area allocation algorithms for minimizing the huge reconfiguration overhead of the current FPGAs. The presented results show that a performance gain of up to 61 %for MPEG2 and 94 % for MJPEG is to be expected when the proposed allocation algorithms are used. In our future work, we will extend the allocation algorithms to take also into account the reconfiguration order of the considered Rops and to exploit parallelism.

References

[1] P. Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Research Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, January 1995.

[2] M. Dales. Managing a reconfigurable processor in a general purpose workstation environment. In *In Proceedings of DATE*, pages 10980–10985, Munich, Germany, 2003.

[3] S. Fekete, E. Khler, and J. Teich. Optimal fpga module placement with temporal precedence constraints. In *Proceedings of DATE*, pages 658–665, 2001.

[4] M. A. George, M. Pink, D. Kearney, and G. Wigley. Efficient allocation of fpga area to multiple users in an operating system for reconfigurable computing. In *Proceedings of ERSA*, pages 238–242, 2002.

[5] R. Maestre, F. J. Kurdahi, N. Bagherzadeh, H. Singh, R. Hermida, and M. Fernndez:. Kernel scheduling in reconfigurable computing. In *Proceedings of DATE*, pages 90–96, 1999.

[6] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *IEEE Transactions on Computers*, 35(2):50–58, February 2002.

[7] E. M. Panainte, K. Bertels, and S. Vassiliadis. Instruction scheduling for dynamic hardware configurations. In *Proceedings of DATE*, pages 100–105, Munich, Germany, March 2005.

[8] Sundance. FC-JPEG04 JPEG Compression Design Specification. pages 1–4, http://www.sundance.com/docs/FC-JPEG04 Sundance - 300504.pdf, 2004.

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

[10] H. Walder and M. Platzner. Online scheduling for block-partitioned reconfigurable devices. In *In Proceedings of DATE*, pages 290–295, Munich, Germany, 2003.

[11] Xilinx Corporation. *Virtex-II Pro Platform FPGA Handbook v2.0*, October 2002.

[12] Xilinx Corporation. *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Functional Description*, June 2004.