Cross-Architectural Design Space Exploration Tool for Reconfigurable Processors

Lars Bauer, Muhammad Shafique, and Jörg Henkel
University of Karlsruhe, Chair for Embedded Systems, Karlsruhe, Germany
{lars.bauer, shafique, henkel} @ informatik.uni-karlsruhe.de

***Abstract*—Processors that deploy fine-grained reconfigurable fabrics to implement application-specific accelerators on-demand obtained significant attention within the last decade. They trade-off the flexibility of general-purpose processors with the performance of application-specific circuits without tailoring the processor towards a specific application domain like Application Specific Instruction Set Processors (ASIPs). Vast amounts of reconfigurable processors have been proposed, differing in multifarious architectural decisions. However, it has always been an open question, which of the proposed concepts is more efficient in certain application and/or parameter scenarios. Various reconfigurable processors were investigated in certain scenarios, but never before a systematic design space exploration across diverse reconfigurable processor concepts has been conducted with the aim to aid a designer of a reconfigurable processor.**

We have developed a first-of-its-kind comprehensive design space exploration tool that allows to systematically explore diverse reconfigurable processors and architectural parameters. Our tool allows presenting the first cross-architectural design space exploration of multiple fine-grained reconfigurable processors on a fair comparable basis. After categorizing fine-grained reconfigurable processors and their relevant parameters, we present our tool and an in-depth analysis of reconfigurable processors within different relevant scenarios.

I. INTRODUCTION

Employing reconfigurable processors [1, 2] affects the application development in two different ways. First, the application-specific hardware accelerators need to be designed for the reconfigurable fabric (typically in a hardware description language) and afterwards, these accelerators need to be operated efficiently. To obtain a benefit from the hardware accelerators the application programmer is required to insert *Special Instructions* (SIs) into the application (e.g. using inline assembly or compiler tools). In a way, both steps are comparable to the development process of *Application Specific Instruction Set Processors* (ASIPs) [3]. However, reconfigurable processors need to determine *which accelerators* should be loaded into the reconfigurable fabric at *which time* during the application execution. This is often accomplished by so-called *prefetching instructions* [4] that trigger the upcoming reconfigurations.

The difference between ASIPs and reconfigurable processors becomes noticeable when executing SIs. An ASIP provides all SIs statically, typically using the same fabrication technology like the core pipeline. A reconfigurable processor instead has to use a different fabrication technology for the SIs (FPGA-like reconfigurable[1]). Additionally, it may not have an SI available when it shall execute, e.g. because the process of reconfiguration has not finished yet. The partitioning of the reconfigurable fabric (e.g how many SIs can be provided at the same time) has a high impact on whether or not an SI is available when it is demanded. Furthermore, architectural parameters (e.g. the reconfiguration time) affect the ability to efficiently provide SIs on demand.

[1] coarse-grained reconfigurable hardware (e.g. ALU array with configurable interconnects) is not the focus of this work

Multifarious processors – varying in their concepts and the complexity of their run-time system – are presented and discussed in literature (see Section II). However, it remains an open question which of the proposed concepts is more appropriate in certain scenarios (e.g. for a specific CPU- and FPGA frequency or the logic capacity of the available reconfigurable fabric etc.). To practically investigate this question it is not adequate to compare the independently published performance values, because they are based on different assumptions, different applications, and different normalizations (i.e. the definition of '1x speedup').

Our contributions are:

- a novel comprehensive design space exploration tool that allows to systematically explore diverse reconfigurable processors through a set of relevant architectural parameters,
- an SI-based categorization of reconfigurable processors, and
- an exploration of the impact of relevant architectural parameters as well as cross-architectural comparisons on a comparable basis conducted with our design space exploration tool.

To the best of our knowledge, this is the first cross-architectural design space exploration of multiple fine-grained reconfigurable processors on a fair comparable basis. Our design space exploration tool provides a valuable aid when developing systems with fine-grained reconfigurable fabrics. We will describe the concepts and advantages/disadvantages of different processor categories in Section III after summarizing the relevant reconfigurable processors from literature in Section II. Furthermore, we will explain the assumptions and realization of our cross-architectural design space exploration tool in Section IV and explain how it can be used to explore the diverse reconfigurable processors. In Section V, we present and analyze cross-architectural comparisons before concluding in Section VI.

II. RELATED WORK

Besides coarse-grained reconfiguration[1] (like ADRES [5]) fine-grained reconfigurable systems can be partitioned into general frameworks and specific processors. General Frameworks like [6, 7] provide a generic environment into which dedicated IP-cores can be loaded on demand. Typically, they are equipped with a general-purpose CPU core that may host an operating system [8] to perform the reconfigurations. Many of these frameworks use the Xilinx FPGAs and tools [9] to realize the reconfiguration in practice.

Some of the early reconfigurable processors like CoMPARE [10] provide a reconfigurable fabric that is coupled to the pipeline of the core processor similar to an ALU and that can be reconfigured to comprise the hardware implementation of a single SI at a time. CHIMAERA [11] instead couples the processor core with a reconfigurable fabric that may contain multiple SIs with at most 9 inputs and 1 output value (register addresses are hardcoded within the SI) at the same time. The MOLEN processor [12] offers an instruction set extension to access a reconfigurable fabric. The instruction set extension can be parameterized to support only one or multiple SIs at a time. The SIs are executed by a general instruction that obtains the address of the configuration bitstream as parameter to identify the demanded SI.

978-3-9810801-5-5/DATE09 © 2009 EDAA

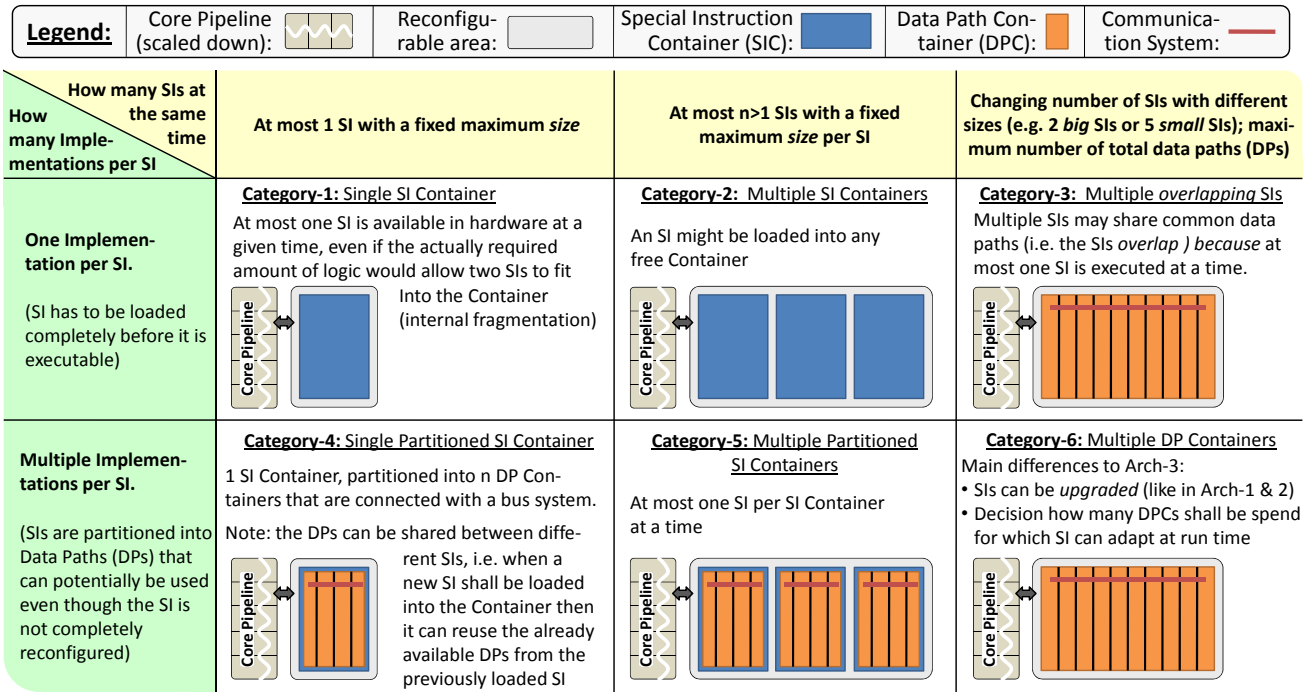

Fig. 1. Classification of Reconfigurable Processors, Differing in the Realization of Special Instructions (SIs)

The RISPP processor introduces a new concept of SIs [13] in conjunction with a run-time system [14] to support them. Each SI exists in multiple implementation alternatives, reaching from a pure software implementation (i.e. without using the reconfigurable fabric) to various hardware implementations (providing different trade-offs between the amount of required hardware and the achieved performance). The concept is to break SIs into elementary reconfigurable data paths that are connected to implement an SI. A run-time system dynamically chooses one alternative out of the provided SI implementations, depending on the current application requirements.

III. Reconfigurable Processor Alternatives

A. SI-Based Categorization of Reconfigurable Processors

We have systematically analyzed the reconfigurable processors presented in Section II and will now present a categorization that covers these processors as well as discloses potential processor candidates that were not yet considered in research prototypes. Besides the later presented architectural parameters (see Section B), the basic distinguishing attribute is the concept to provide SI implementations, i.e. *'how many SIs may be available at the same time?'* and *'how many implementation alternatives exist per SI?'*. We assume that all SIs may also be executed using the Instruction Set Architecture of the core pipeline to bridge the SI reconfiguration time. This is typically realized using a simple trap mechanism or a special branch instruction that jumps to a handler when the SI is not available in hardware. Furthermore, all processors have certain similarities due to technical constraints of available reconfigurable fabrics and their tools. The implementation of a hardware description into a reconfigurable fabric always features a rectangular outline, as this is the typical shape into which the 'place & route' tools can place the implementation. Furthermore, these rectangular SI implementations cannot be placed at each arbitrary position within the reconfigurable fabric. To establish the communication between the CPU and the SI, dedicated communication ports are provided at fixed locations within the reconfigurable fabric. To connect an SI implementation to these communication ports, the rectangular SI implementation needs to be aligned to these communication ports. This leads to the notion of *SI Containers* (SIC), i.e. a rectangular region in the reconfigurable fabric that provides a communication port to the SI implementation that is currently loaded into this container.

Category-1: Single SI Container Early investigated reconfigurable processors (e.g. [10]) provide exactly one SIC as shown in Fig. 1. This SIC has a fixed size of logic that is determined while designing the system. If the SIC is too small, then not all potential SIs fit within. Instead, if it is too big (i.e. providing more logic resources than what is actually required or beneficial), then this has a negative effect on the hardware utilization and the time it takes to reconfigure the entire SIC. Additionally, large SICs come with the problem of internal fragmentation, i.e. the available reconfigurable fabric can only be utilized to implement one SI at a given time although the sole amount of reconfigurable fabric might be sufficient to implement more.

Category-2: Multiple SI Containers The extension of Category-1 partitions the reconfigurable fabric into multiple SICs such that potentially multiple SIs are available at the same time (see Fig. 1). For a given logic capacity of the reconfigurable fabric, the question arises whether to partition it into few rather big SICs or into more rather small SICs (depending on the requirements of the targeted applications etc.). Prominent examples of this category are [11, 12].

Category-3: Multiple Overlapping SIs The major conceptual difference of Category-3 compared to Category-1 and 2 is the partitioning of the reconfigurable fabric. Instead of partitioning it into SICs, it is partitioned into Data Path Containers (DPCs) as shown in Fig. 1. This modification is based on the observation that SIs are sometimes composed out of more elementary computations, e.g. *transform* operations like DCT, inverse DCT, or (inverse) *Hadamard Transformation* (HT) require some computational butterflies that have similarities. Therefore, when e.g. requiring the four SIs for (inverse) DCT and (inverse) HT within a loop, it may be more beneficial to load just one transform butterfly into the reconfigurable fabric and to share it among the four different SIs (i.e. the SI implementations *overlap*, because they

share some DPCs). It is also noticeable that the amount of SIs that may be available in the DPCs is not fixed. It actually depends on the SIs and their capability of reusing data paths. Using the concept of SICs (Category-1 and 2) for providing the above-mentioned four SIs in the same loop, four SICs have to be available and the butterfly functionality has to be loaded into each of them. Thus, further reconfigurable fabric is required.

However, to partition the reconfigurable fabric into DPCs, these DPCs have to be able to communicate with each other to eventually realize the functionality of the SI. Therefore, a communication system (e.g. [15]) needs to connect the DPCs. On the one hand, this communication system might affect the scalability of the system, while on the other hand the actually required logic capacity of the reconfigurable fabric might be reduced, depending on the potential to share data paths between the SIs.

Category-4: Single Partitioned SI Container On the one hand, this category combines Category-1 and 3, i.e. one SIC is internally partitioned into multiple DPCs. On the other hand, this category introduces the new concept of multiple different implementations per SI. This extension is based on the observation that an SI often requires multiple instances of a certain data path. E.g., the DCT SI internally executes the transform butterfly multiple times, depending on the size of the input data on which the DCT shall execute. Depending on the logic capacity of the reconfigurable fabric, either multiple instances of the butterfly can be loaded into different DPCs to be executed in parallel or one instance is sequentially executed multiple times to eventually achieve the same functionality. This leads to an SI implementation with less area requirements at the cost of reduced performance.

The concept of providing multiple implementations per SI and partitioning the SI into data paths allows for *upgrading* the currently available SI implementation by loading further data paths. Thus *upgrading* allows moving the execution towards a more parallel scheme. The concept of *upgrading* potentially reduces the problem of a long reconfiguration time. For instance, Category-1 and 2 only provide a single implementation per SI and the complete SIC has to be reconfigured to load it. Thus, bigger SICs automatically increase the reconfiguration time.

Category-5: Multiple Partitioned SI Containers This category is the combination of Category-2 and 3 (see Fig. 1), i.e. it can provide multiple SIs at the same time and each SI can be *upgraded*, as the SICs are partitioned into DPCs. Compared to Category-3 it solves the scalability problem (i.e. the inter-DPC communication bus does not cover all DPCs but only those within one SIC). This is accomplished by fixing the number of supported SIs at design time and thus it requires specific knowledge of the application requirements.

Category-6: Multiple Data-Path Containers Combining the advantages of Category-3 and 5, Category 6 provides a homogeneous region of DPCs which may be used to implement one rather *big* SI (i.e. using most of the DPCs) or multiple *small* SIs. This decision can be re-evaluated whenever the SI is demanded and thus the system can adapt to changing run-time requirements like changing SI execution frequencies in the application or different SI demands of different applications in a multi-tasking environment. Category-3 instead is just able to provide one fixed implementation per SI whereas Category-5 is limited to the number of SIs it can support, thus both are lacking adaptivity (like required in a multi-tasking scenario). The advantage of the greater flexibility comes at the disadvantage of potential scaling problems for the inter-DPC communication system (similar to Category-3) as well as the additional overhead of a run-time system that actually makes the decisions on how to deploy the provided flexibility depending upon the current system state.

Category-1, 2, and 6 were actually investigated in literature [10-15]. However, Category-3 and 5 are advantageous as well since they reduce the overhead compared to Category-6:

- Category-3 solves to potential fragmentation problem of Category-1 and 2 without introducing the need to maintain multiple implementation alternatives per SI as in Category-6.
- Category-5 solves the problem of the rather long reconfiguration time of Category-1 and 2 (i.e. until the *full* SI is reconfigured) without demanding the central bus system of Category-6 that connects all DPCs with each other.

B. Relevant Architectural Parameters

Besides different concepts of SI-implementations for the reconfigurable fabric, further parameters and properties of the surrounding system are relevant for evaluating the performance. Table I shows the parameters that our design space exploration tool supports, covering all performance-wise relevant parameters.

TABLE I. ARCHITECTURAL PARAMETERS

Parameter	Symbol	Physical Unit
CPU frequency	f_{CPU}	[MHz]
FPGA frequency	f_{FPGA}	[MHz]
Reconfiguration bandwidth	R	[MB/s]
#Memory ports	P	N/A
Bit width per port	W	[Bits]

- The **Operating Frequency** of the FPGA and the core pipeline may differ. In general, the non-reconfigurable fabric (core pipeline) can run with a faster frequency than the reconfigurable fabric (FPGA) due to the different fabrication technologies.
- The **Reconfiguration bandwidth** determines the time it takes to reconfigure parts of the FPGA (e.g. an SIC or DPC). The reconfigurable fabric is reconfigured by loading new configurations (i.e. bitstreams). The size of these bitstreams and the reconfiguration bandwidth determine the reconfiguration time.
- The **number of memory ports** (that can access data memory in parallel at independent addresses) and the **bit width per memory port** determine the amount of input data that is available to perform computations. Note that only SIs that execute on the reconfigurable fabric may use the extended memory access. To be compliant to the ISA, the core pipeline always uses one memory port with 32-bit width.

IV. OUR DESIGN SPACE EXPLORATION TOOL

To be able to simulate/evaluate the diverse reconfigurable processors and relevant architectural parameters described in Section III, we designed and implemented a flexible design space exploration tool supporting a wide set of configurable parameters. Note that we do not intend to automatically explore the design space, but rather we provide an accurate and configurable simulator with a corresponding tool chain that allows comparing reconfigurable processors on a fair basis and investigating the impact of diverse architectural parameters. Fig. 2 visualizes the major components and interactions of our simulator as a UML class diagram. It is partitioned into three major parts: the core pipeline and run-time system (managing the application execution and reconfigurations), the SIs (representing their implementations, execution times, etc.), and the FPGA. The FPGA is an abstract class that can be instantiated as a SIC FPGA (for Category-1, 2, 4, 5 in Fig. 1) or as a DPC FPGA (for Category-3 and 6). Note that the SICs can contain zero (Category-1 and 2) or multiple (Category-4, 5) DPCs internally. Correspondingly, each DPC belongs to zero (Category-3 and 6) or one (Category-4 and 5) SIC.

The SIs are composed of at least two implementations: one using the data paths and one using the Instruction Set Architecture (ISA) of the core pipeline. The SIs and their composition are determined by an external XML file that defines all required information like the name of the SI, the instruction format and opcode

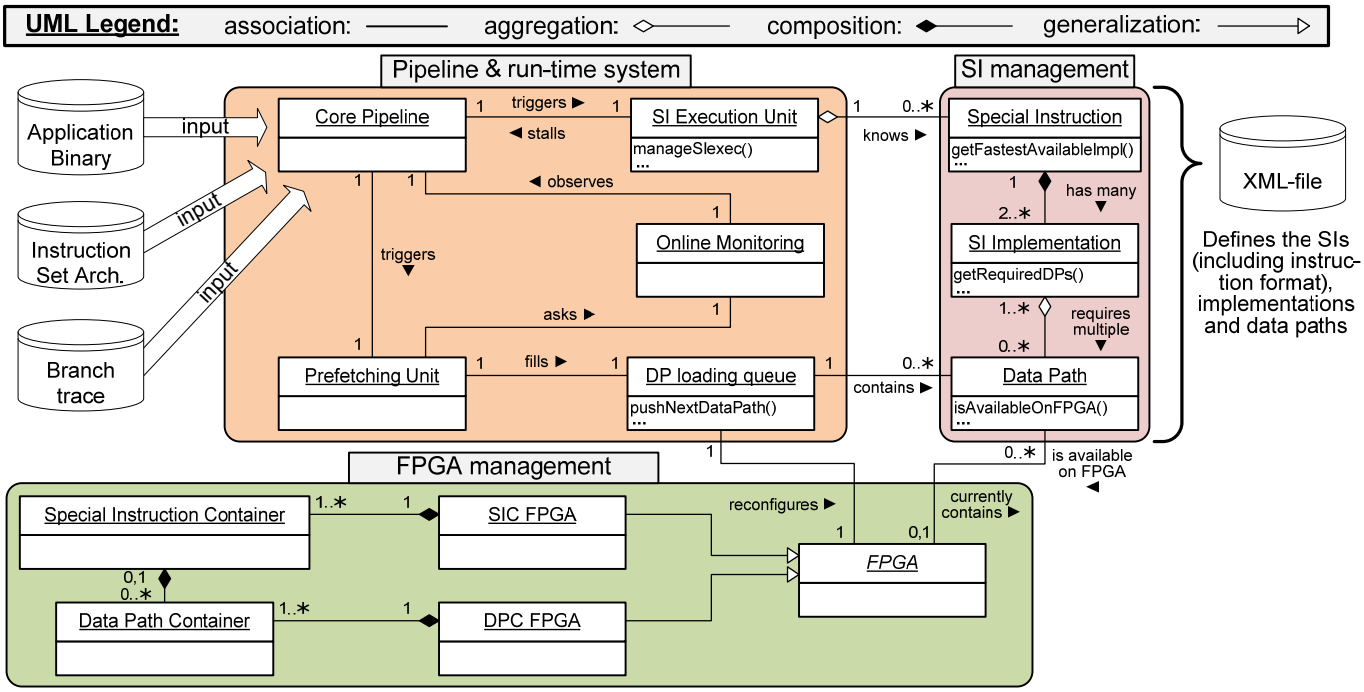

Fig. 2. Internal Composition of our Design Space Exploration Tool, showing Module Interactions

(to be able to decode it out of the application binary) and the available implementations with the latency and data path requirements. Furthermore, the data paths contain the information on the bitstream size (to determine the reconfiguration time for Category-3 to 6) and the logic requirements (to determine how many fit into an SIC for Category-1 and 2). The XML file is identical for all processors, as it provides all possible SI implementations. When the simulation starts, the set of all possible implementations is automatically restricted, corresponding to the constraints of the selected processor category. For instance, when only one hardware implementation shall be available per SI, then the fastest implementation that fits into the SICs is selected statically. However, for different data memory ports and bit widths, different XML files exist. They are semi-automatically created as part of our tool chain, taking the actual data-flow graph of the SI (the data paths correspond to the nodes in the graph) with the corresponding load/store activities as input and then scheduling it, i.e. determining a starting time for each data path. This schedule is automatically performed for all possible resource constraints (i.e. how many instances of a certain data path are available at the same time) to obtain all implementation alternatives.

The simulation of the pipeline receives the application binary along with the ISA as input. The ISA is automatically extended by the SI information from the XML file to be able to decode all instructions. The non-SI instructions are actually not executed by the pipeline, but a branch trace (containing all taken branches in their actually executed sequence) is provided to mimic the exact application execution. This allows abstracting our simulator from the actual ISA[2] and reduces the simulation time, as we are interested in investigating different processor categories and architectural parameters for one fixed branch trace. The branch trace is derived from an instruction set simulator (ISS). We have assured that the execution in our simulator for reconfigurable processors matches the actual execution. Therefore, we have modeled the pipeline in our simulator and provided the information which instruction requires how many cycles in which pipeline stage (e.g., the *mult* instruction stalls the execution stage). However, the register file and data memory accesses are not simulated, thus, although at any time we know which instruction is currently in which pipeline stage, we cannot determine the current content of the register file or the actual data memory accesses. To simulate the exact data memory accesses (to e.g. attach a cache simulator) an additional data access trace needs to be provided from the ISS. Currently, each load/store instruction is configured to require two cycles. When the pipeline issues these memory accesses then the cycle time corresponds to the CPU frequency, otherwise to the FPGA frequency. This mimics the scenario that all accesses to the instruction memory and stack are cache hits and that all accesses to the data input stream are handled by a scratchpad memory. This allows us to investigate the differences of the processor category without affecting the results by cache effects.

The online monitoring in Fig. 2 is initialized with offline profiling data that is derived from the ISS and that is updated at run time. The relevant information that is demanded by the prefetching unit is the expected SI execution frequency. For Category-6 this information is used to select implementations for the demanded SIs and thus to determine the DPC-distribution among the SIs. The simulator creates a detailed log file containing information about the current system state and the decisions made. The planned reconfigurations are printed along with the current state of the FPGA and the information, which SI implementation is currently available, e.g. in which SIC. Furthermore, statistics on the SI executions are shown, e.g. which SI implementations were executed since simulation start or in the recent time.

V. Design Space Exploration

In the following, we use our design space exploration tool to investigate the categorized reconfigurable processors and architectural parameters from Section III. Our goal is to examine the architectural parameters as well as performing a cross-architectural comparison on a comparable and fair basis. To be able to investigate the advantages and drawbacks of the processor categories it is mandatory to benchmark them with applications that actually challenge them. For instance, many of the typical ASIP standard benchmarks from MiBench [16] and MediaBench [17] are rather

[2] it is currently setup for MIPS and SPARC-V8; the SI instruction format and opcode for both ISAs are provided in the XML file

small application kernels that often contain only one inner loop that corresponds to the major hot spot. For a reconfigurable processor this means that only the first execution of this loop actually demands a reconfiguration, because afterwards the application just executes the same loop with the same SIs again.

We decided to perform an in-depth analysis of an H.264 video encoder as a detailed case study instead of summarizing the results of multiple applications from MiBench or MediaBench[3]. H.264 is a rather complex application with three subsequent outer loops (i.e. *Motion Estimation*, *Encoding Engine*, and *Loop Filter*) that are executed for each input frame. They contain multiple application kernels and demand run-time reconfiguration to adapt the hardware towards the currently executing outer loop. Table II summarizes relevant parameters of the SIs and DPs that we have implemented for this application. The bitstream sizes and logic requirements for the DPs were taken from actual hardware implementations to adapt our simulator to the running FPGA prototype. It requires 10.6 seconds to execute the application (encoding 20 frames in QCIF resolution, i.e. 176x144) on the corresponding GPP (i.e. a Sparc-V8 without hardware accelerators) at 100 MHz and 2.1 seconds at 500 MHz. Altogether, we have performed 4620 different simulation and will now present relevant results for the different categories and parameters.

TABLE II. PROPERTIES OF THE SIS AND DPS FOR OUR BENCHMARK

Parameter	Value	Comment
# SIs	9	4/1 in/out register (e.g. for mem. addresses)
# Data Paths	10	2/2 32-bit in/out values
SI composition	1 - 4 DPs	Utilizing multiple instances per DP
SI memory accesses	0 – 128 words	For some SIs the input from register file is sufficient, others work on data memory
DP Bitstream	42,719 - 43,638 Byte	Bitstream for partial reconfiguration on Xilinx Virtex-II xc2v6000 FPGA
DP logic requirements	16 – 192 slices	Note: these readings correspond to the pure computational logic without the necessary interconnection overhead

Category-1 and 2: We first investigate Category-2 (including Category-1 as a special case) regarding its behavior for varying amount of SICs. Fig. 3 shows how the available reconfigurable fabric may be partitioned into SICs, providing either more SICs (different lines) or more reconfigurable fabric per SIC. It is noticeable that the number of SICs does not change the performance when the SICs are only one CLB[4] wide. Many SI implementations do not fit into a one CLB-wide reconfigurable fabric. Thus, they cannot be provided in this case. When only one SIC is available, setting its size to be larger than 4 does not improve the performance further because the sequential execution of the SIs that cannot be executed in the reconfigurable fabric (because only one SIC is available) limits the performance.

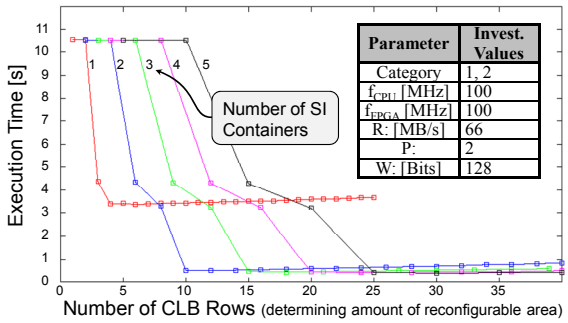

Fig. 3. Category-1 and 2: Partitioning the Hardware into SICs

Category-4 and 5: In Fig. 4, we present a design space exploration for Category-5 (multiple partitioned SICs; including Category-4 as a special case). Altogether, 2016 different simulations for this category are summarized. It is noticeable that the number of available SICs and DPCs are dominating the other readings, i.e. they determine the basic shape of the results. Again, we can see that one SIC is not sufficient for this application. It is noticeable that Category-5 does not suffer as much from *overdesigned* SICs (i.e. too many DPCs per SIC, resulting in a rather long reconfiguration time) as Category-2, because not necessarily the whole SIC but only the required DPCs need to be reconfigured.

Category-3, 5, and 6 with different frequencies: Here, we investigate the impact of the individual parameters and we cross-compare different processor categories to each other. Fig. 5 investigates the impact of different CPU- and FPGA frequencies for Category-3, 5, and 6. For Category-5, we decided to provide two SICs, because from Fig. 4 we learned that two SICs achieved the best performance for less than eight DPCs and for more than eight DPCs it still achieved a better or similar performance (compared to three or more SICs). As seen, all processor categories require a critical amount of DPCs to cover the major hot spots. Category-3 and 6 require only five DPCs to reach this point (see Fig. 5) because they are not limited in the decision of how many SIs to realize with the given amount of DPCs. For instance, they can spend all available DPCs to implement the two SIs for *Motion Estimation* and the single SI for *Loop Filter* respectively, while supporting 3-6 SIs in the *Encoding Engine* (depending on the amount of available DPCs).

When less DPCs than the critical amount are provided, then the CPU frequency has a noticeable impact on the overall execution time of all processor categories. When providing a sufficient amount of DPCs, the FPGA frequency has the larger impact. For instance in Category-6 with f_{FPGA}=100 MHZ, changing f_{CPU} from 100 to 500 MHz results in 3.28x performance improvement when utilizing 3 DPCs but only 1.23x for 20 DPCs. Instead, for f_{CPU}=500 MHz, changing f_{FPGA} from 50 to 100 MHz results in 1.09x for 3 DPCs, but 1.86x for 20 DPCs. As long as not all major hot spots are covered by a hardware implementation, they are executed rather sequential (i.e. by the core pipeline) and therefore the CPU frequency has the higher impact. After the critical point, these hot spots are executed in a parallel manner on the FPGA and therefore the FPGA parameters are more relevant.

For the FPGA the amount of parallelism also depends on the amount of input data that is available for the DPCs. While the readings in Fig. 5 were obtained using two memory ports with 128 bits each, the readings in Table III summarize the behavior when only one 32-bit port is available. It is noticeable that the FPGA frequency has a much higher impact than before because now the computation on the FPGA is performed in a more sequential way (although still more parallel than on the core pipeline), because insufficient input data is available.

TABLE III. AVERAGE EXECUTION TIME (BETWEEN 14 AND 24 DPCS) WHEN ONLY PROVIDING 1 MEMORY PORT WITH 32 BIT

Category	f_{FPGA} [MHz]	f_{CPU}=100 MHz	f_{CPU}=200 MHz	f_{CPU}=500 MHz
3	**50**	1.1699 s	1.1334 s	1.0880 s
5	**50**	1.2389 s	1.1575 s	1.0732 s
6	**50**	1.1286 s	1.0999 s	1.0487 s
3	**100**	0.6520 s	0.6186 s	0.5840 s
5	**100**	0.7125 s	0.6220 s	0.5738 s
6	**100**	0.6032 s	0.5760 s	0.5526 s

Memory Settings: To analyze the impact of the memory settings further, Table IV shows the simulation results when varying the amount of memory ports and their bit widths. It is noticeable that the settings with two ports (e.g. 2*32 bits) always outperform the

[3] note that many of their sole application hot spots (e.g. DCT, SAD, VLC, or FIR) are components of the H.264 encoder

[4] Configurable Logic Block: determining the amount of reconfigurable logic on the FPGA

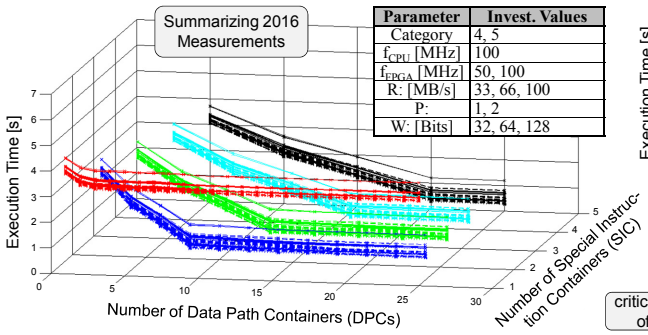

Fig. 4. Exploring Different Parameters for Category-4 and 5

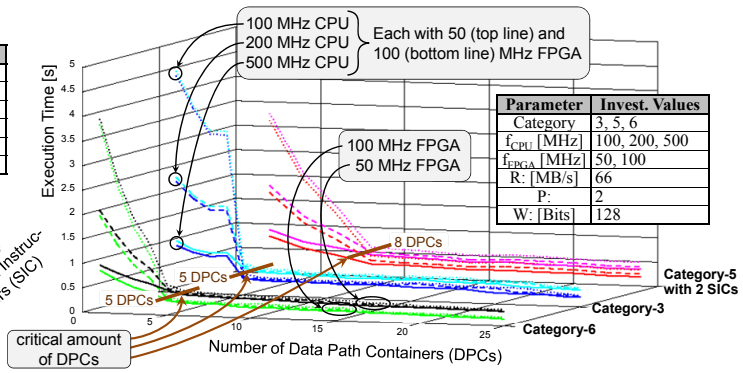

Fig. 5. Cross-Architectural Comparison (Category-3, 5, and 6) for Different CPU and FPGA Frequencies

corresponding single-port settings (i.e. providing same total bit width; e.g. 1*64). Improving from 1*32 to 2*32 results in a 1.68x speedup, but improving from 2*64 to 2*128 only achieves 1.15x.

TABLE IV. AVERAGE EXECUTION TIME (BETWEEN 14 AND 24 DPCS) FOR CATEGORY-6 USING DIFFERENT MEMORY PARAMETERS[a]

# Memory Ports	Port bit width	Average Exec. Time
1	32	0.6032 s
2	32	0.3594 s
1	64	0.4165 s
2	64	0.2703 s
1	128	0.3244 s
2	128	0.2347 s

[a]: f_{CPU}=100 MHz, f_{FPGA}=100 MHz, R=66MB/s

Reconfiguration Bandwidth: Finally, we investigate the impact of different reconfiguration bandwidths on the overall performance. Fig. 6 shows how the execution time decreases for increased bandwidth. For clarity, we have omitted the readings with less than four DPCs because they are significantly slower due to insufficient logic capacity of the reconfigurable fabric (as shown in Fig. 5). Instead, we want to investigate the importance of the bandwidth when all other bottlenecks (e.g. insufficient input data or insufficient DPCs etc.) are removed. As shown, a rather slow bandwidth (1-15 MB/s) can be compensated by a higher amount of DPCs and vice versa, a low amount of DPCs (5-15) can be compensated by a faster bandwidth. For 25 DPCs, increasing the bandwidth from 25 MB/s to 66 MB/s only leads to a speedup of 1.06x. However, for 10 DPCs a speedup of 1.26x can be obtained when increasing to 66 MB/s.

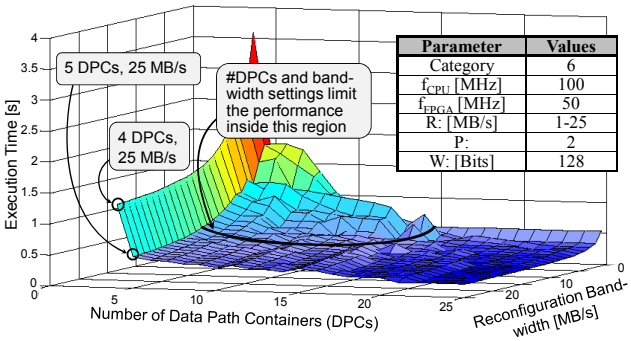

Fig. 6. Impact of the Reconfiguration Bandwidth

VI. CONCLUSION

We have presented a novel comprehensive design space exploration tool with a wide set of parameters. We used it to investigate the impact of relevant architectural parameters as well as to perform cross-architectural comparisons on a fair comparable basis. By categorizing existing reconfigurable processors, we unveiled further – up to now not-yet-considered – processors (i.e. Category-3, 4, and 5) and provided an initial evaluation of them. Summarizing our results, we conclude that Category-6 provides the highest performance and the highest flexibility. However, it is not the focus of our simulator to estimate the necessary overhead for the run-time system to support this processor category. If the high flexibility or performance is not required in a certain scenario, then Category-3 and 5 may be good candidates to trade-off performance vs. overhead. Our design space exploration tool presents an important design aid for system designers dealing with reconfigurable processors, as it allows a systematically cross-architectural design-space exploration of multiple fine-grained reconfigurable processors on a comparable basis.

REFERENCES

[1] S. Vassiliadis, D. Soudris, "Fine- and Coarse-Grain Reconfigurable Computing", Springer, ISBN 978-1-4020-6504-0, 2007.

[2] C. Bobda, "Introduction to Reconfigurable Computing", Springer, ISBN 978-1-4020-6088-5, 2007.

[3] J. Henkel, "Closing the SoC Design Gap", IEEE Computer, vol. 36, issue 9, pp. 119-121, 2003.

[4] Z. Li, S. Hauck, "Configuration prefetching techniques for partial reconfigurable coprocessor with relocation and defragmentation", Int'l Symp. on FPGAs, pp. 187-195, 2002.

[5] F. Bouwens, et al., "Architectural Exploration of the ADRES Coarse-Grained Reconfigurable Array", ARC, pp. 1-13, 2007.

[6] M. Ullmann, et al., "An FPGA Run-Time System for Dynamical On-Demand Reconfiguration", Int'l Parallel and Distributed Processing Symposium (IPDPS), pp. 135-142, 2004.

[7] M. Majer, J. Teich, A. Ahmadinia, C. Bobda, "The Erlangen Slot Machine: A Dynamically Reconfigurable FPGA-Based Computer", VLSI Signal Processing Systems, pp. 15-31, 2007.

[8] E. Lübbers, M. Platzner, "ReconOS: An RTOS supporting Hard- and Software Threads", FPL, pp. 441-446, 2007.

[9] P. Lysaght, B. Blodget, J. Mason, J. Young, B. Bridgeford, "Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration on Xilinx FPGAs", FPL, pp. 1-6, 2006.

[10] S. Sawitzki, A. Gratz, R. G. Spallek, "CoMPARE: A Simple Reconfigurable Processor Architecture Exploiting Instruction Level Parallelism", Conf. on Parallel and RT Systems, pp. 213-224, 1998.

[11] Z. A. Ye et al., "CHIMAERA: a High Performance Architecture with a tightly coupled reconfigurable functional unit", Intl. Symp. on Computer Architecture (ISCA), pp. 225-235, 2000.

[12] S. Vassiliadis, et al., "The MOLEN polymorphic processor", Trans. on Computers, vol. 53, issue 11, pp. 1363-1375, 2004.

[13] L. Bauer, M. Shafique, S. Kramer, J. Henkel, "RISPP: Rotating Instruction Set Processing Platform", Design Automation Conference (DAC), pp. 791-796, 2007.

[14] L. Bauer, M. Shafique, S. Kreutz, J. Henkel, "Run-time System for an Extensible Embedded Processor with Dynamic Instruction Set", Conf. on Design, Automation & Test (DATE), pp. 752-757, 2008.

[15] L. Bauer, M. Shafique, J. Henkel, "A Computation- and Communication-Infrastructure for Modular Special Instructions in a Dynamically Reconfigurable Processor", FPL, pp. 203-208, 2008.

[16] MiBench (http://www.eecs.umich.edu/mibench/).

[17] C. Lee, M. Potkonjak, W. H. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems", MICRO, pp. 330-335, 1997.