Tuning Coarse-Grained Reconfigurable Architectures towards an Application Domain

Julio Oliveira Filho, Thomas Schweizer, Tobias Oppold, Tommy Kuhn, Wolfgang Rosenstiel
Department of Computing Engineering
University Tübingen, Germany
{oliveira,tschweiz,oppold,kuhn,rosenstiel}@informatik.uni-tuebingen.de

Abstract
Design decisions, such as type and ratio of functional units, strongly determine the later flexibility of domain-specific FPGAs and coarse-grained dynamically reconfigurable arrays. For that reason the design of such reconfigurable architectures is done considering a set of target applications. However, it is not yet clear how to simultaneously consider all applications to determine an appropriate distribution of resources. In this paper, we deal with this problem through a design exploration methodology that exposes how each application contends in the use of coarse-grained resources. Our method is used to assess the adequacy of one given resource distribution answering two intermediary questions: 1) how evenly does it addresses all applications in the domain, and 2) how flexible remains the architecture on achieving diverse optimization goals.

1. Introduction

At the last decade, there was an increasing interest from industry and academia in coarse-grained reconfigurable architectures [1]. State-of-the-art FPGAs, such as Xilinx-Virtex Series [2], include hardwired memory blocks and specific arithmetic functional units. These custom-build modules avoid the natural performance penalties due to spreading many small logical resources across a too general routing network. Concurrently, several dynamically reconfigurable architectures, such as DRP [3], Silicon Hive [4] and ADRES [5], have been proposed as an array of coarse-grained word-wide processing elements. These architectures were shown to exhibit improved performance and power reduction when targeted towards an application domain.

On fine-grained architectures, functional modules are composed at application-mapping phase by connecting several small lookup tables and simple gate blocks. However, coarse-grained architectures are made up of fixed circuitry modules determined at design time and incorporated in the architecture when the device is manufactured. We call that phase the configuration phase. Therefore, hardwired memory blocks, specialized arithmetic units and other custom-made modules cannot be easily reconfigured to execute other functionalities. Their incorporation implies a reduction in the device flexibility.

Still though, if the intended range of applications for a particular reconfigurable device is known or can be characterized at design time, these modules may be incorporated to the device while providing adequate flexibility to accommodate the applications’ requirements. Thus, the configuration phase must be accomplished considering the set of applications that fulfills the system purpose (domain). Design decisions, such as type and ratio of modules to be included, strongly influence the later usage of the device. This paper is focused on the configuration phase of coarse-grained reconfigurable architectures. It approaches the problem on how to simultaneously consider all applications of the domain in order to determine an adequate resource distribution.

Figure 1. CRC Model

In order to illustrate the nature of that problem, we make use of the Configurable Reconfigurable Core (CRC) model, developed in [6]. The CRC is a general model for processor-like reconfigurable architectures and is depicted in Figure 1. It consists of a two-dimensional array of processing elements (PEs), surrounded by an...
interconnect network. Each PE contains local registers, a word-wide arithmetic and logic functional unit and a context memory. Parameters of the model describe the array geometry, i.e. number of PEs (width and length), number of entries in the context memory, number of registers, and type of functional unit and interconnect network. Configuring the parameters determines an architecture instance that can be synthesized. Applications may then be mapped and simulated. The term processor-like reconfigurability [7] alludes to the ability of the architecture to perform reconfiguration within one clock cycle. At the beginning of each clock cycle, an entry of the context memory is selected by a control unit that implements the finite state machine.

The authors mapped four applications from image processing domain (AND, XOR, Spatial Convolution and Matrix Multiplication) into three different architecture instances. These instances were obtained by adjusting the following parameters of the CRC model: number of processing elements (PEs) and number of entries in the context memory (contexts). Other parameters were kept constant. The ratio between PEs and contexts was set up so that each one of the three architectures has the same approximate area of $700,000\text{um}^2$. Figure 2 depicts the execution time measured in control cycles and normalized using the first instance (8 PEs, 32 contexts) as the reference architecture. When the distribution of resources is 12 PEs and 16 contexts (second instance) almost all applications had a speed up because of the augmented number of processing elements. However, the best achieved performance for the XOR application was smaller than in the first instance. That happened because we could not apply the same loop unrolling factor due to the smaller number of available contexts. On a third instance variant (17 PEs, 8 contexts) the spatial convolution mapping could not be done due to too few available contexts.

The previous example shows that some applications of the domain are improved by the redistribution of resources, while others get worse or even are excluded. That problem is especially crucial for coarse-grained architectures because, as discussed previously, a successful design should offer overall performance gain while maintaining flexibility to accommodate all applications.

We present a methodology to carry out the design space exploration exposing how each application compete on type and quantity of resources to be allocated. In order to accomplish that task, we describe our target architecture as parametrizable model, such as the CRC. Then parameters that correspond to the allocation of architectural resources, such as number of PEs and number of contexts, are used as optimality criteria on a multi-dimensional Pareto analysis. A mapping is Pareto-optimal if it cannot be realized using less resources of one type without demanding more resources of any other type. Usual optimality objectives such as performance and power may be simultaneously considered. We show that the proposed method may answer two important questions necessary to assessing the adequacy of the architecture to a given domain: 1) given a possible resource allocation, how evenly does it address all applications in the domain, and 2) how flexible is it on achieving diverse optimization goals?

![Figure 2. Distribution of resources impacts differently each application in domain.](image)

The remainder of this paper is organized as follows. In the next section we discuss the problem of assessing the adequacy of coarse-grained architectures to a domain. In section 3, we present how the design space exploration should be carried in order to allow such assessment. Section 4 presents our results on characterizing the design space for a parametrizable architecture model toward an image processing application set. Section 5 and 6 resumes our future work and conclusions.

2. Related Work

The problem of exploring the design space for coarse-grained architectures has been addressed in literature, especially for System-on-Chip and platform-based architectures. For a good review of approaches at several abstraction levels, we indicate [8]. However, several questions remain open when the objective is tuning domain-specific FPGAs, and dynamically reconfigurable coarse-grained architectures, such as the
ones discussed in Section 1, towards an application domain.

Ken Eguro and Scott Hauck [9] concentrate on the problem of functional unit allocation for domain-specific FPGAs – determining the most appropriate quantity and ratio of functional units and routing resources. The authors present three techniques to allocate functional units that attempt to balance performance and area constraints on domains that have vastly different hardware requirements. As start point for all presented techniques, the authors consider a feasible architecture that meets minimal requirements. This architecture is then iteratively altered through the addition and subtraction of functional units. At each iteration, the applications are mapped to the architecture with a new functional unit distribution and evaluated according to a cost function. The techniques differentiate from each other on the considered constraints: the first technique is a performance-constrained algorithm and the other two use simulated annealing in an area-constrained approach. This design space exploration was shown to lead to good results. However, the number of iterations is very high, which implies a long development time. Additionally, it does not expose how each application contend on the distribution of functional units, making it hard to the designer to know in which direction the design should be refined at each iteration.

Our approach deals with the problem in the same level of abstraction, however, we concentrate in dynamically reconfigurable architectures that are built as an array of coarse-grained processing elements. Additionally, we simultaneously analyse several architecture instances, and at each iteration, new design points are generated based on the information gathered so far. Finally, our method explicitly exposes the demand for architectural resources at each application, providing useful information to be used on the generation of new instances.

The work presented in [10] proposes a multi-dimensional Pareto analysis and a design space exploration flow similar to the one used here, that is based on a space of parameters. It focuses on platform-based systems and system-on-chip. We differentiate in the choice of the objective criteria set: they consider power, area and performance while we characterize the area through the detailed description of ratio and type of custom-made modules.

In the next section, we introduce our approach to collect and organize information during the design space exploration phase. Such information simultaneously exposes how each application uses the available hardware resources. We call that method Design Space Characterization.

3. Design Space Characterization

Figure 3 depicts our flow. The input comprehends a representation of the application domain and a parametrizable architecture model. The application domain is defined through a set of applications or algorithms that fulfill the system purpose. Such assumption is extensive enough to enclose modern conceptions of application domains. It does not rely on common characteristics of applications or limit them to specific classes, such as “pipelinable”. Therefore, even instances with very different attributes and requirements are similarly accepted. The disadvantage of defining the domain through a set of applications is that later the designed architecture may be inefficient in supporting unforeseen applications.

![Figure 3. Design Space Characterization Flow](image)

The architecture to be designed is captured through a parametrizable model, so that one specific parametrization determines an architecture instance. In doing so, we force the possible architectural solution to be within the parametrization space. Such assumption is necessary to restrict the number of possible variants if the architecture should be developed from scratch [10]. This work focuses on assessing a possible architecture instance. We do not discuss the adequacy of the model itself to the application domain.

The Design Space Characterization is a proposal on how to conduct the design exploration phase. It consists in three major activities: Transformation space Exploration, Application Mapping, Mapping Exploration and Design-space Analysis. The flow in Figure 3 is depicted in a straightforward version because, at each phase, a major number of possible mapping solutions are evaluated for each application of the domain. Nevertheless, it may also be executed iteratively, where at each iteration includes new mapping possibilities for each application, but still considering the previously one. We get in details for each phase.
3.1 Transformation Space Exploration

The Transformation Space Exploration phase uses as input all C-described applications in the definition set of the domain. We call the original description of one application base description. Objective of this phase is to reveal several code variations obtained through transformation of the base description. These variations arise from applying high-level source code-transformations/optimizations, such as loop unrolling or common sub-expression elimination. Exploring several possibilities is important in two aspects. First, high level code transformations may influence the demand for resources, as well as the system performance, in divergent ways. For example, loop unrolling technique generally leads to better execution times, but due to the parallelism augmentation, it increases the demand for more functional units, storage elements and power. Second, it may ease to suit one architecture instance to equitably support the domain.

We represent the transformation space with a graph, as depicted in Figure 3(a). Nodes designate a possible description variant. An edge indicates which transformation was used to convert one variant (input node) into another (output node). The sequence of transformations, including type and order of execution, may be automatically performed by a compiler or interactively hinted by the designer. The compiler approach produce a better cover of the transformation space, however it may lead to a combinatorial explosion. That problem may be avoided using the second method. Here, the designer points which transformations should be executed by the compiler after analysing one particular variant.

When this phase is completed, each application in the definition set is represented by a graph with its base description and respective transformations. The extended definition set is much more representative than its original version. It discloses several potential improvements of the base description toward several optimization goals and architectural resource requirements.

3.2 Application Mapping Exploration

The input for the mapping exploration consists on the extended definition set produced in the previous step. All nodes in each transformation graph are considered individually as follows.

Each description is mapped in several architecture instances. The architecture instances are obtained by adjusting the parameters of the architecture model. There are two approaches for executing the parametrization: pre-definition and fractional. The pre-definition approach consists in fixating several instances before the beginning of the mapping exploration. In that case, the applications are not contemplated when setting the parameters. The objective in this approach is to produce instances with differing resource distributions. For example, Figure 2 depicts three architecture instances with same average area but pre-parametrized with different PE vs. context ratios. Such approach is interesting especially if an overall area constraint is set. The fractional approach consists on a partial parametrization of the instance. Some parameters are pre-defined while others will be evaluated during the mapping process. For example, the designer may fixate the number of PEs and evaluate, during the mapping, how many contexts where necessary in order to achieve a certain performance. That approach is indicated if constraints for performance and power consumption are known. It has also the advantage that information about the real demand for a particular resource will be included on the parametrization. We recommend the use of both approaches. Therefore, fixed pre-defined and fractioned architecture instances constitute the target for the mapping process.

Additionally, our methodology lets an open space in this step to consider several mapping strategies. For example, it may be interesting to map the description variants of one particular application using software pipelining techniques to improve throughput. Meanwhile in another application a particular scheduling method may produce the best usage for storage elements. Although that is a very powerful and important aspect, the complexity of the Application Mapping Exploration phase increases significantly. Therefore, without loss of generality to our analysis, this paper will not consider the diversity on mapping strategies.

Figure 3(b) depicts the mapping activities. Each description is mapped in all architecture instances. Mappings in pre-defined instances receive the parameters of the model as hardware constraints and should optimize another metric of interest, such as execution time. Mappings on fractioned instances consider the pre-adjusted parameters and maybe performance as constraint, while trying to optimize the usage of loose parameters. After the mapping exploration, all necessary information to characterize the design space is ready. It is now necessary to organize it properly.

3.3 Design Space Analysis

Traditional design space exploration methodologies summarize a possible design through multiple evaluation metrics such as cost, performance and power. We call these evaluation metrics objective criteria because each metric may be represented by a totally ordered set of
We define as objective criteria vector the vector \( \mathbf{O} = (P_1, P_2, \ldots, P_n) \), where \( P_i, i \in 1,2,\ldots,n \) are parameters of the model with possible values \( p_i \) over a totally ordered set \( \Omega_i \). \( P_i \) can also be an evaluation metric for power or performance, and thus \( p_i \) is respectively a value for the consumed power or for performance such as execution time or throughput. For the cases where a parameter is used, they may reflect directly in the objective criteria vector. For example, number of contexts is an objective criterion because it may be represented in \( \mathbb{N}^+ \) and smaller number of contexts is always preferred to larger ones. Some parameters such as qualitative ones may require their value-set to be mapped through a function into an ordered set. In a third possibility, a given parameter cannot be mapped into an ordered set with an associated preference order. In that case, that parameter cannot be supported by this approach.

We substitute the area metric by an objective criteria vector with all parameters that represent architectural resources of interest. Therefore, our evaluation metrics are power, performance and the elements in the objective criteria vector. Once considered that extension, we can proceed with a traditional multi-dimensional Pareto optimality analysis. A mapping is Pareto optimal if no other mapping is at least as good with respect to every evaluation criterion and better with respect to at least one evaluation criterion. The Pareto set is the set of all Pareto-optimal designs in the design space.

Each Pareto point in our analysis is optimal in at least one aspect: the usage of one resource, performance or power consumption. Substituting the simple metric by an objective criteria vector exposes the trade-off between architectural resources within one application. Although that trade-off cannot be directly compared between applications, we show in the next section that a metric for it can be developed.

In Figure 3(c), we show two possible instances of the objective criteria vector \( \mathbf{O} = (\# \text{PEs}, \# \text{Contexts}, \text{Execution Time}) \), respectively \( A = (17,8,1000) \) and \( B = (12,8,1000) \). The mapping represented by \( A \) is not Pareto-optimal, because \( B \) is better than it in the usage of PEs without prejudice of any other criterion.

### 4. Results on Architecture Adequacy

We applied our approach to characterize a subset of the parameterization space of the CRC Architecture Model toward a set of image processing applications. The definition set representing our application domain was composed with 6 base descriptions of real-world algorithms for image processing:

- **Histogram Equalization** [11] – used to stretch the contrast of an image by uniformly redistributing the grey values. Loop with pixel colour transformation was considered.
- **Binary AND and XOR Composition** [12] – typical for superposition and subtraction of images. Composition uses simultaneously two images to generate a composite.
- **Spatial Discrete Convolution 3x3 and 5x5** [11] [13] – typical method for smoothing and sharpening of images with 3x3 and 5x5 masks. Loop for applying the mask was considered.
- **Matrix Multiplication** [14] – several algorithms for matrix multiplication were included in this application. They constitute small examples.

We restricted the parametrization space to only two parameters of the model: number of PEs and number of entries in the context memory. These two elements are responsible for most part of the area occupation on a real architecture instance [6]. Additionally, the trade off between these two parameters highly determines the use of reconfigurability in the final device. A small number of processing elements containing a great number of context entries implies that computation will be mostly sequential and reconfigurability will extensively used. Otherwise, a great number of processing elements with less available context entries will benefit the spatially mapped operations in detriment of several sequential reconfigurations. All other parameters represented an unlimited resource amount. For example, PEs have always registers available if necessary. Similarly, the interconnect network was assumed to provide enough resources so that all mappings could be considered feasible. These assumptions affect conclusions inferring real complete instances. The impact of such restrictions is further discussed in section 5.

Table 1 shows the information used when executing our design space characterization method. The Transformation step was performed using the indicated set of code transformations in an assisted basis. For each application, it was defined a subset of transformations that led to code reduction or parallelism augmentation. During the Mapping Exploration step, there were considered fractioned architecture instances with number of PEs varying between 2 and 19. The number of necessary contexts in order to obtain the best performance was annotated after the mapping. The number of analysed mappings for each application is indicated in the fourth column. The objective criteria vector, extracted in each case, comprises the number of PEs in the target architecture instance, the number of...
used contexts and the execution time (in control cycles) necessary to process one 320x320 grey scale picture. We do not consider power here.

<table>
<thead>
<tr>
<th>Application</th>
<th>Transformations</th>
<th># Investigated Mappings</th>
<th># Pareto-optimal Mappings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Histogram Equalization</td>
<td>• Dead Code Elimination</td>
<td>882</td>
<td>42 (4.7%)</td>
</tr>
<tr>
<td></td>
<td>• Tree height reduction</td>
<td>711 – AND</td>
<td>42 (5.9%)</td>
</tr>
<tr>
<td></td>
<td>• Loop Unrolling (2x – 16x)</td>
<td>1080 – XOR</td>
<td>49 (4.5%)</td>
</tr>
<tr>
<td>Spatial Discrete Convolution</td>
<td>• Common Sub-Expression Elimination</td>
<td>1080</td>
<td>58 (5.3%)</td>
</tr>
<tr>
<td>Matrix Multiplication</td>
<td>• Constant Folding</td>
<td>440</td>
<td>10 (2.2%)</td>
</tr>
</tbody>
</table>

The Pareto-sets were isolated for each application. Table 1 also depicts the proportion of Pareto points in relation to the number of total investigated mappings. In all cases, the number of Pareto-optimal mappings is much smaller. That happened because all description variants were mapped in all architecture instances, even when it obviously was a bad solution. For example, description variants with high degree of parallelism were insisted to be mapped into instances with small number of PEs. That augments the need for contexts without real gain in performance. Such aspect indicates that an iterative execution of the flow, where only some few mappings are allowed per application, may lead to a more efficient exploration.

Given one architecture instance, we say that it satisfies a point in the design space if its parametrization attends completely the objective criteria vector of that point. In other words, the demand for architectural resources is fulfilled by the architecture instance. Additionally, given one architecture instance δ and one application α we define $C_δ(α) = n$, where n is the cardinality of the Pareto-set in the design space of α that are satisfied by δ.

$C_δ(α)$ is a metric on how flexible the architecture instance δ attends the mapping of application α toward multiple optimality goals. This results directly from the fact that each Pareto point is optimal in at least one possible criterion. At last, we define the cover of an architecture instance δ for a domain A, and indicate $C^A_δ$, as $C^A_δ = \sum_{α \in A} C_δ(α)$. $C^A_δ$ is a metric of the adequacy of the architecture instance to the domain. It measures how many optimal designs may be achieved considering all applications in the domain.

We used both metrics to analyze the problem previously discussed in section 2. Results are depicted in Figure 4. Each bar indicates the cover of one possible trade-off between PEs and contexts in the parameterization. The balanced instance with 12 PEs and 16 Contexts supports about 41% more optimal mappings as the first instance, and up to 70% more when compared to the third. That last instance is highly prejudiced by the fact that no spatial convolution designs could be implemented. Within each bar, $C_δ(α)$ is outlined for each application. It shows that the second proposal also allows more Pareto-points to be found within each application. In resume, the second instance has the most adequate distribution of resources to attend this domain.

### 5. Future Work

Our work is not yet so far in order to indicate which could be the best distribution of resources, basically because of the huge complexity involved on the design space exploration (DSE). In order to obtain a more precise analysis of real CRC instances it is necessary to include other parameters in the objective criteria vector. Other important parameters to be considered are number of registers per PE, number of external memory modules, number of I/O ports, and type of interconnect network. That happens because restricting these resources to a
small number do co-influence the results for performance and number of contexts we present here.

Up to now, all description variants are being mapped in all architecture instances, even when it leads to bad mappings. An algorithm to prune the design space should be developed.

Additionally, the conclusions of our proposal are still restricted to assessing the adequacy of a given parametrization (architecture instance) to the domain. It cannot yet indicate the optimal parametrization. Our future efforts concentrate on achieving these goals.

6. Conclusions

We present a proposal to carry out the design space exploration when targeting coarse-grained domain-specific reconfigurable architectures. Our methodology improves the state of the art in two important directions. First, it clearly exposes the competition for architectural resources within and among applications. That information is crucial to obtain an architecture that evenly addresses all the applications in the domain. Second, based on our analysis we develop a metric to assess the adequacy of an architecture instance to the domain. It constitutes a measure of how many optimal designs may be implemented in that instance. Two problems are solved using that metric: 1) given several possible resource allocations, which one is the most adequate to the domain; 2) given one application and one possible resource allocation, how flexible the architecture is in achieving multiple mapping goals.

Finally, in order to validate our methodology, we characterize the parametrization space of the CRC architecture model toward an image processing application domain.

8. Acknowledgments

This work is supported by Programme Alßan, European Union Programme of High Level Scholarships for Latin America, n. E04D045457BR and DFG Priority Program 1148 on Reconfigurable Computing Systems, RO 1030/13 – 1 and RO 1030/13 – 2.

7. References