R. Hartenstein, M. Herz, Th. Hoffmann, U. Nageldinger: Generation of Design Suggestions for Coarse-Grain Reconfigurable Architectures; 10th International Workshop on Field Programmable Logic and Applications, FPL '2000, Villach, Austria, Aug.27-30, 2000

Generation of Design Suggestions for Coarse-Grain Reconfigurable Architectures

R. Hartenstein, M. Herz, Th. Hoffmann, U. Nageldinger

Computer Structures Group (Rechnerstrukturen), Informatik

University of Kaiserslautern, D-67653 Kaiserslautern, Germany

hartenst@rhrk.uni-kl.de - http://xputers.informatik.uni-kl.de - Fax: +49 631 205 2640

Abstract. Coarse-grain reconfigurable architectures have been a matter of intense research in the last few years. They promise to be more adequate for computational tasks due to their better efficiency and bigger speed. As the coarse granularity implies also a reduction of flexibility, a universal architecture seems to be hardly feasible. Based on the KressArray architecture family, a design-space exploration system is being implemented, which supports the designer in finding an appropriate architecture for a given application domain. By analysing the results of a number of different experimental mappings, the system derives suggestions how the architecture can be enhanced. This paper presents an analyser tool, which allows the generation of such suggestions using approximate reasoning based on fuzzy logic. The tool is flexible enough to support different data gathering methods and an extensible rule set.

1. Introduction

While early solutions in reconfigurable computing were based on fine-grained FPGAs, in the recent years also coarse-grain reconfigurable architectures have been developed, which are capable of implementing high-level operators in their processing elements, featuring multiple-bit wide datapath: [1], [2], [3], [4], [5], [6], [7]. The coarse-grain approach promises advantages over FPGA-like fine-grain architectures:

- Less configuration data is needed, since there are typically orders of magnitude less operators than in an FPGA.
- Routing structures transfer words instead of bits, thus needing less control signals and associated configuration storage. The lower amount of configuration data supports fast loading and dynamic partial reconfiguration.
- Coarse-grain architectures are much more area-efficient. This applies to both, operators and routing resources. Powerful operators like ALUs, MACs and others can be directly implemented in silicon, instead of being mapped onto zillions of CLBs surrounded by wide areas of fine grain routing resources.
- The coarse granularity is better suited for application development of computational tasks from a high-level language and similar sources.

To obtain sufficient flexibility for high production volume most future SoC implementations need some percentage of coarse grain reconfigurable circuitry [8]. Reconfigurable computing even has the potential to question the dominance of the microprocessor [9] [10]. However, for the use of a coarse granularity some problems have to be solved to cope with the reduced flexibility compared to FPGA-based solutions:

- Processing elements of coarse-grain architectures are more "expensive" than the logic blocks of an FPGA. While it is possible for FPGA mappings, that a number of logic blocks is unused or cannot be reached by the routing resources, especially the latter situation is quite annoying for coarse-grain architectures due to the fewer number of processing elements.
- While the multi-bit datapath applies well to operators from high-level languages, operations working on smaller word-lengths and especially bit manipulation operations are weakly supported. If such operations occur, like e.g. in data compression algorithms, they may result in a complex implementation requiring several processing elements.

Due to these problems, the selection of architectural properties like datapath width, routing resources and operator repertory is a general problem in the design of coarse-grain architectures. Thus, a design space exploration is done in many cases, to determine suitable architectural properties. As the requirements to the architecture are mostly dependent on the applications to be mapped onto it, previous efforts use normally a set of example applications, e.g. DES encryption, DCT, or FIR filters. For examples such an explorations is described in [7] to determine the best bit width for ALUs of the architecture. According to this method, coarse-grain architectures are often targeted for specific application domains, like multimedia [2], [7] or DSP [4].

To find a suitable architecture for a given application domain, the KressArray Xplorer framework is being implemented at Kaiserslautern University [11], [12]. The framework uses the KressArray [5] architecture family as basis for an interactive exploration process. When a suitable architecture has been found, the mapping of the application is provided directly. The designer is supported during the exploration by suggestions of the system how the current architecture may be enhanced. This paper focuses on the tool which gives these suggestions.

The rest of this paper is structured as follows: The next section briefly sketches the KressArray architecture family. Then, the design space for the exploration process is described. In section 4, the general approach for an interactive design space exploration for an application domain is presented. In the following section, some selected related work is sketched briefly. After this, a short overview on the KressArray Xplorer framework is given. The next section motivates our approach for the generation of design suggestions, followed by an overview on the implementation of this approach. Finally, the paper is concluded.

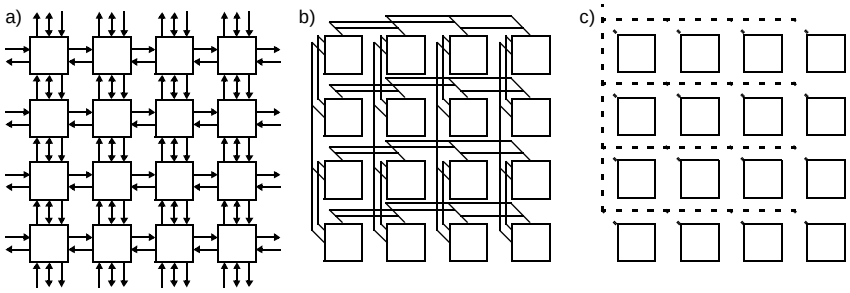

Figure 1: Three levels of interconnect for KressArray architectures: a) nearest neighbor links, b) backbuses in each row or column (for more examples see figure 2), c) serial global bus.

2. The KressArray architecture family

The KressArray family is based on the original mesh-connected (no extra routing areas, see figure 2 d, e) KressArray-1 (aka rDPA) architecture published elsewhere [5], [13]. An architecture of the KressArray family is a regular array of coarse grain reconfigurable DataPath Units (rDPUs), each featuring a multiple-bit datapath and providing a set of coarse grain operators. The original KressArray-1 architecture provided a datapath of 32 bits and all integer operators of C, the proposed system can handle also other datapath widths and operator repertories. The different types of communication resources are illustrated in figure 1. There are three levels of interconnect: First, a rDPU can be connected via nearest neighbor links to its four neighbors to the north, east, south and west. There are unidirectional and bidirectional links. The data transfer direction of the bidirectional ones is determined at configuration time. Second, there may be backbuses in each row or column, which connect several rDPUs. These buses may be segmented, forming several independent buses. Third, all rDPUs are connected by one single global bus, which allows only serial data transfers. This type of connection makes only sense for coarse grain architectures with a relatively low number of elements. However, a global bus effectively avoids the situation, that a mapping fails due to lack of routing resources. The

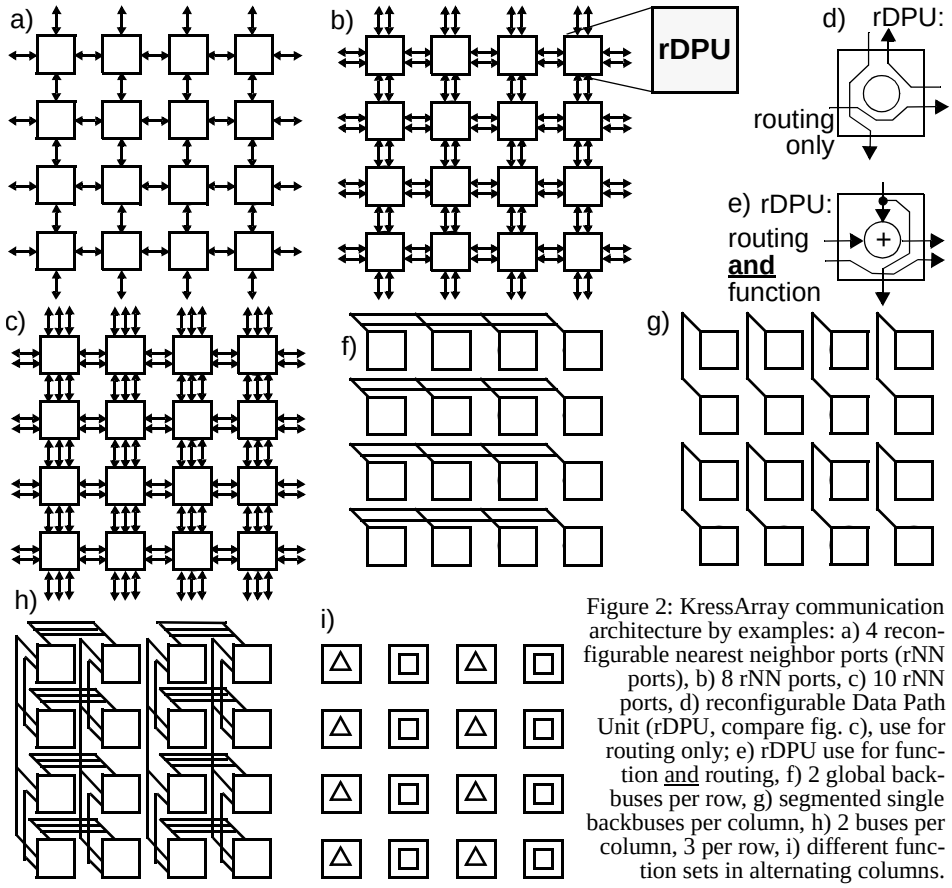

Figure 2: KressArray communication architecture by examples: a) 4 reconfigurable nearest neighbor ports (rNN ports), b) 8 rNN ports, c) 10 rNN ports, d) reconfigurable Data Path Unit (rDPU, compare fig. c), use for routing only; e) rDPU use for function and routing, f) 2 global backbuses per row, g) segmented single backbuses per column, h) 2 buses per column, 3 per row, i) different function sets in alternating columns.

rDPUs themselves can serve as pure routing elements, as an operator, or as an operator with additional routing paths going through. Some more communication architecture examples are shown in figure 2 [14].

The number and type of the routing resources as well as the operator repertory are subject of change during the exploration process. Typically, a trade-off has to be found between the estimated silicon area and the performance of the architecture. Examples for trade-off issues are sequential or parallel multiplication or the realization of complex operators like multiply-accumulate by one or multiple rDPUs.

3. The KressArray Design Space

The KressArray structure defines an architecture class rather than a single architecture. The class members differ mainly by the available communication resources and the operator repertory. Both issues have obviously a considerable impact on the performance of the architecture. In the following, we define the design space for KressArray architectures based on the introduction given in section 2.

The following aspects of a KressArray architecture are subject to the exploration process and can be modified by the tools of the exploration framework:

- The size of the array.
- The operator repertory of the rDPUs.
- The available repertory of nearest neighbor connections. The numbers of horizontal and vertical connections can be specified individually for each side and in any combination of unidirectional or bidirectional links.
- The torus structure of the array. This can be specified separately for each nearest neighbor connection. The possible options are no torus structure or torus connection to the same, next or previous row or column respectively.
- The available repertory of row and column buses. Here, the number of buses is specified as well as properties for each single bus: The number of segments, the maximal number of writers, and the length of the first segment, which allows buses having the same length but spanning different parts of the array. Areas with different rDPU functionality. For example, a complex operator may be available only in specific parts of the array. This allows also the inclusion of special devices in the architecture, like embedded memories. The operator repertory can be set for arbitrary areas of the array, using generic patterns described by few parameters.
- The maximum length of routing paths for nearest neighbor connections. This property is relevant for timing restrictions.
- The number of routing paths through a rDPU. A routing path is a connection from an input to an output through a rDPU, which is used to pass data to another rDPU.
- The interfacing architecture for the array. Basically, data words to and from the KressArray can be transferred by either of three ways: Over the serial global bus, over the edges of the array, or over an rDPU inside the array, where the latter possibility is mostly used for library elements.

In order to find a suitable set of these properties for a given application domain, an interactive framework is currently developed. The framework, called KressArray Xplorer,

allows the user a guided design of a KressArray optimized for a specified problem. At the end of the design process, a description of the resulting architecture is generated.

4. Design Space Exploration

The global approach to design space exploration for a domain of several applications is shown in figure 3. First, all applications are compiled into a representation in an intermediate format, which contains the expression trees of the applications. All intermediate files are analyzed to determine basic architecture requirements like the number of operators needed. The main design space exploration cycle is interactive and meant to be performed on a single application. This application is selected from the set of applications in a way, that the optimized architecture for the selected application will also satisfy the other applications. This selection process is done by the user, with a suggestion from the system. For our current example setup to optimize the performance by exploring the communication architecture, we use the application with the worst regularity, as this is supposed to have the highest routing requirements. An approach to measure the regularity of an application has been published in [17].

The exploration itself is an interactive process, which is supported by suggestions to the designer, how the current architecture could be modified. The application is first mapped onto the current architecture. The generated mapping is then analyzed and statistic data is generated. Based on this data, suggestions for architecture improvement are created using a fuzzy-logic based approach described below. The designer may then chose to apply the suggestion, propose a different modification, return to a previous design version, or end the exploration. Some modifications allow the new architecture to be used directly for the next mapping step, while others will require a reevaluation of the basic architecture requirements and/or a reselection of the application for the exploration. Given our setup for performance optimization, a change to the routing resources should not require a reevaluation, as the current application is supposed to be the one with the worst routing requirements. In contrast to this, a change of the operator repertory for the rDPUs requires the replacement of subtrees in the dataflow graph, thus effecting the number of required rDPUs in the array as well as the complexity of all applications. Thus, for a change of the operator repertory, a reevaluation is required.

After the exploration cycle has ended, the final architecture has to be verified by mapping the remaining applications onto it. On the one hand, this step produces mappings of all applications to be used for implementation, while on the other hand, it is made sure that the architecture will satisfy the requirements of the whole domain.

5. Related work

For most reconfigurable architectures, the designers did a thorough evaluation of different architectures. In [7], target application properties are considered to determine the datapath width for processing elements for the PipeRench architecture. The results are gained by evaluating several example applications mapped onto the architecture. In [15], Moritz et al. present a framework for architectural design space exploration of processors for the Raw Machine [6], considering application requirements. This approach is based on mathematical models, which have to be provided for the applications to be considered.

Due to the switched interconnect of Raw, this work does not address requirements of routing resources.

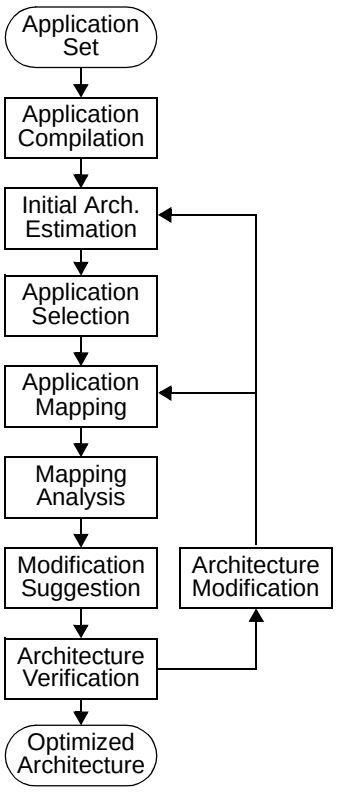

Figure 3: Global approach for domain-specific architecture optimization.

Other design exploration frameworks are found mostly in the area of VLSI design. One approach which comes closest to ours is described in [16]. The framework by Guerra et al. allows interactive design space exploration for datapath-intensive ASICs, starting from a high level description. This work also considers application properties like regularity, which has an impact on the routing requirements. Design guidance is generated, by polynomial functions and parameterized rules. However, we find the way the suggestions are generated not adequate for our purposes, as we expect the properties of the design, which are extracted not to be that clear. This is because of the use of heuristic algorithms on the one hand, and the possibility to explore heterogeneous structures with areas of different operator repertories on the other hand, as both may distort the characterizations extracted from the design. Further, the approach described allows optimization for a single application, while we target optimization for application domains consisting of several applications.

6.The KressArray Xplorer

This section will give a brief description of the components of the KressArray Xplorer, which is published elsewhere [12]. An overview on the Xplorer is shown in figure 4. The framework is based on a design system, which can handle multiple KressArray architectures. It consists of a compiler for the high-level language ALE-X, a scheduler for performance estimation, and a simulated-annealing based mapper. This system works on an intermediate file format, which contains the net list of the application, delay parameters for performance estimation, the architecture description, and the mapping information. The latter is added by the mapper at the end of the synthesis process. An architecture estimator determines the minimum architecture requirements in terms of operator numbers and suggests the application with the biggest complexity to an interactive graphical user interface. This interface is also generally used to control all other tools. Further, it contains two interactive editors, an architecture editor, which allows to change the architecture independently by the design suggestions, and a mapping editor, which allows to fine-tune the result of the mapper. An analyzer generates suggestions for architecture improvements by information gathered from the mapping and other sources. An instruction mapper allows the change of the operator repertory by exchanging complex operators in the expression tree with multi-operator implementations. A simulator allows both simulation of the application on the architecture as well as generation of a behavioral

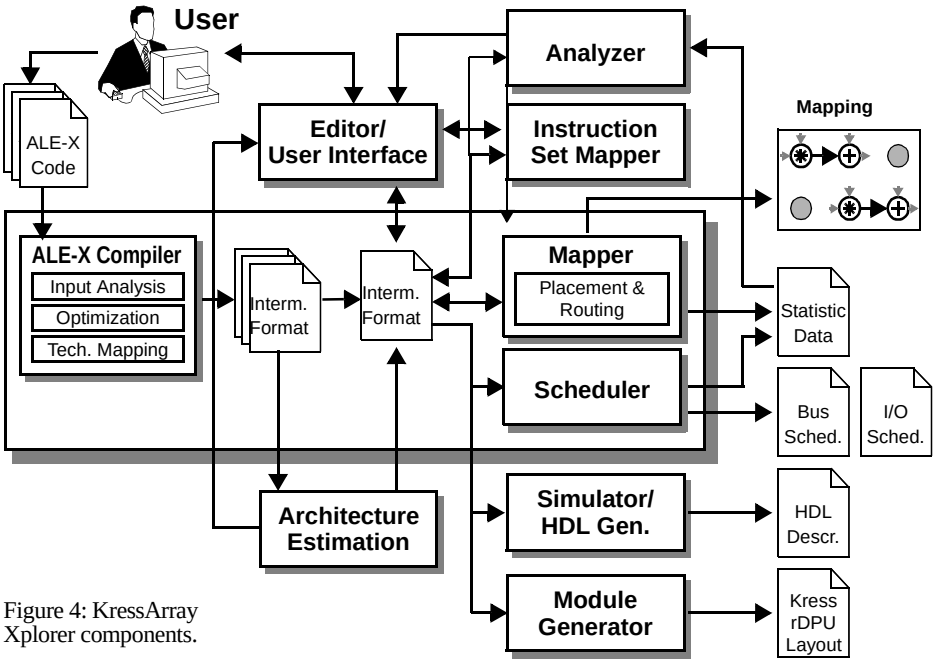

Figure 4: KressArray Xplorer components.

HDL (currently Verilog) description of the KressArray. Finally, a Module Generator (planned) can be used to generate the final layout of a KressArray cell.

7. Generation of Design Suggestions

In this chapter, we will discuss the problem of the generation of design suggestions for our framework and motivate our approach, which is performed by the analyzer tool of the Xplorer framework. The problem of the generation of feedback on a given design can be split into several subproblems:

- Analysis of the current architecture and gathering of information. This includes the combination of data to gain derived information.
- Generation of suggestions from this information.
- Ranking of the suggestions after their importance.

In our approach, the basis for the information gathering step is the mapping produced by the design system. Data gathered includes the usage of buses and nearest-neighbor connections in percent, the number of serial bus connections, the estimated number of execution cycles, the estimated area, and others. Other properties are taken from the current application set, like the maximum tree height of the expression trees and the maximum estimated graph complexity. Some of these values are generated from more basic properties, e.g. the nearest neighbor connection usage is calculated from the number of used connections divided by the total number of connections.

The generation of suggestions needs a kind of reasoning, which is based on expert knowledge. This knowledge, can be described using implication rules of the form „IF (property x applies) THEN (action y would be appropriate)“.

Some of the information in the knowledge can be directly expressed as ’sharp’ values, e.g. the clause "(area > budget_threshold)" describes the quality, if an area constraint is met. However, there are also measures, which are affected with a certain inexactness. For example, the usage of the routing resources is highly dependent on the quality of the placement and routing and may change from mapping to mapping due to the heuristic algorithm used. A clause like "(routing_resource_usage > threshold)" might produce different results in subsequent runs of the mapper. To overcome these problems, we chose to model the properties in our approach by linguistic variables [18] and use fuzzy reasoning [19] to generate the design feedback. This way of reasoning has shown to give good results for other areas like stock-market advice. As the concept of fuzzy logic is widely known, we will only give a very brief informal sketch of the issues relevant for our system.

A *linguistic variable* describes a measure not by its numerical value, but by natural language terms, called *linguistic values*. E.g. for the routing resource usage the linguistic values could be ‚low‘, ‚mediocre‘, ‚high‘, and ‚full‘. A measured value (in percent) of the usage is then transformed by describing, how much the numerical value applies to each linguistic value, by giving an according score between 0 and 1. E. g. a usage of 70% may lie between ’mediocre’ and ’high’, which is expressed by assigning a score of 0.4 to ’mediocre’ and 0.6 to ’high’ and 0 to the other values. The translation from the numerical value to the scores is typically done by *membership functions*.

By employing these concepts, it is possible to overcome the problems with uncertainty and describe knowledge in a more natural way, e.g.:
"IF (routing_resource_usage is ’high’) AND (performance is ’low’) AND
(number_of_back_buses is ’low’) AND (average_fan_out is ’high’) THEN
(add_back_bus=’high’)"

Note, that rules with ’sharp’ values can also be modeled by this approach as a special case. Ranking of different suggestions can be included into the rules themselves, as the resulting suggestion is also a linguistic value with an associated score. For the final presentation of the suggestions to the user, the linguistic variables can directly be used.

8. Implementation of the Analyser Tool

The main tasks of the analyzer tool, according to the discussion in the previous section, are the gathering of information and the generation of suggestions using the methods described above. The gathering itself requires some flexibility, as there are several sources for the data. The current intermediate file containing the mapping is the main source, but other data, like performance estimation, is produced by other programs. Furthermore, both the ruleset as well as the information database have to be extensible. To manage all this, a plug-in concept has been chosen, which allows maximal flexibility. The general structure of the analyzer with example plug-ins is shown in figure 5.

The analyzer tool itself contains the data gathered by the plug-ins. The information gathered is supposed to be either numeric values or fuzzy sets. Such data can be dynamically generated by the analyzer and referenced by other plug-ins using a unique name. An exception are the expression tree, the architecture information, and the mapping, which are complex data structures created by analysis of the intermediate file. On start of the

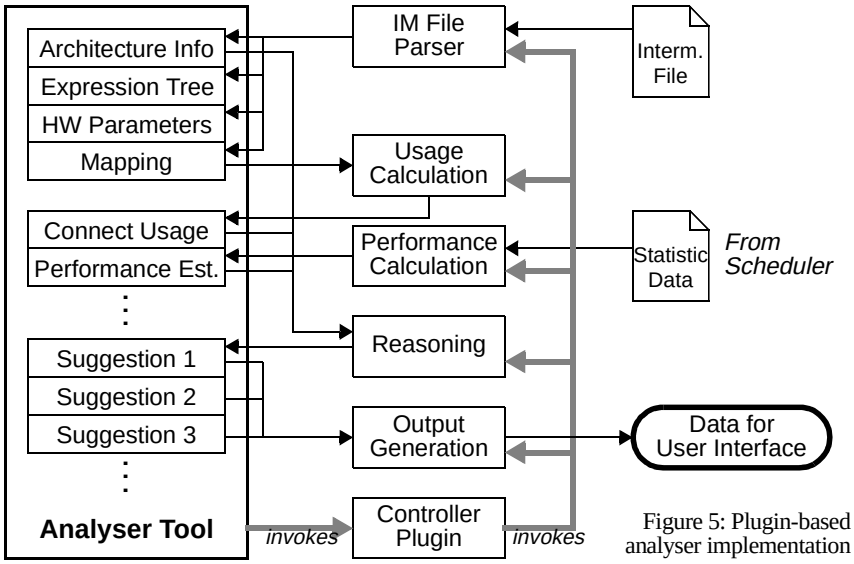

Figure 5: Plugin-based analyser implementation

analyzer, the name of a plug-in is passed as argument, which is then invoked first to run the analysis process.

The plug-ins are loaded dynamically and can access all data stored at the analyzer. Basically, each plug-in has the capability to invoke other plug-ins as well as external programs. In both cases, arguments can be passed to the child plug-in or process. However, we considered it more convenient to employ dedicated controller plug-ins, which are invoked first by the analyzer and start the other plug-ins needed for the analysis. Such a controller plug-in is analogous to a shell script, except that it is included into the plug-in concept. The first plug-in invoked is the parser plug-in, which reads the intermediate file and builds the complex data structures in the analyzers. Following plug-ins work on these data structures to extract more information, like the usage of the routing connections or the average fan-out of the operators. Other plug-ins add to the database by running external programs or analyzing files from other programs, like the statistics from the scheduler, which contain a performance estimation.

As the fuzzy logic operations in the reasoning plug-in involve more complex computations, there is only one plug-in for all rules. The rule to be applied and the required values are passed as arguments to this plug-in. Typically, a whole ruleset is applied, which leads to several suggestions, with each suggestion being affected with an according ranking. After the ruleset has been applied, an output plug-in collects the suggestions and sorts them after their ranking. Then, they are brought into a HTML-like file format to be presented to the user by the User Interface.

9. Conclusions

An interactive approach for the design-space exploration of mesh-based reconfigurable architectures from the KressArray family has been presented. An according framework called KressArray Xplorer is based on a design system which allows the specification of the input language in a high-level language. During the exploration pro-

cess, which is based on iterative refinement of the current architecture, the designer is supported by suggestions on how the current solution may be improved. An analyzer tool has been presented, which makes such suggestions by approximate reasoning employing fuzzy logic modeling. By using a plug-in-based approach, the tool is flexible enough to allow both the addition of new architecture properties to be considered and the extension of the rule set.

Literature

1. E. Mirsky, A. DeHon: „MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources“, Proc. FPGAs for Custom Computing Machines, pp. 157-166, IEEE CS Press, Los Alamitos, CA, U.S.A., 1996.
2. A. Marshall et al.: A Reconfigurable Arithmetic Array for Multimedia Applications; FPGA'99, Int'l Symposium on Field Programmable Gate Arrays, Monterey, CA, U.S.A., Febr. 21 - 23, 1999
3. C. Ebeling, D. Cronquist, P. Franklin: „RaPiD: Reconfigurable Pipelined Datapath“, Int'l Workshop on Field Programmable Logic and Applications, FPL'96, Darmstadt, Germany, Sept 1996.
4. R. A. Bittner, P. M. Athanas and M. D. Musgrove: „Colt: An Experiment in Wormhole Run-time Reconfiguration“; SPIE Photonics East `96, Boston, MA, USA, November 1996.
5. R. Kress: „A Fast Reconfigurable ALUs for Xputers“, Ph.D. thesis, Univ. Kaiserslautern, 1996.
6. E. Waingold et al.: „Baring it all to Software: Raw Machines“, IEEE Computer 30, pp. 86-93.
7. S. C. Goldstein, H. Schmit, et al.: „PipeRench: A Coprocessor for Streaming Multimedia Acceleration“; Int'l Symposium on Computer Architecture 1999, Atlanta, GA, USA, May 1999.
8. J. Rabaey. “Low-Power Silicon Architectures for Wireless Communications”; Embedded Tutorial, ASP-DAC 2000, Yokohama, Japan, Jan. 2000
9. R. Hartenstein: “Der Mikroprozessor im Neuen Jahrtausend”; Elektronik, Januar 1000
10. R. Hartenstein (invited paper): The Microprocessor is no more General Purpose: why Future Reconfigurable Platforms will win; Int'l Conf. on Innovative Systems in Silicon, ISIS'97, Austin, Texas, USA, Oct 1997
11. R. Hartenstein, M. Herz, Th. Hoffmann, U. Nageldinger: „Mapping Applications onto Reconfigurable KressArrays“; International Workshop on Field Programmable Logic and Applications, FPL'99, Glasgow, Scotland, August/September 1999.
12. R. Hartenstein, M. Herz, Th. Hoffmann, U. Nageldinger: „Synthesis and Domain-specific Optimization of KressArray-based Reconfigurable Computing Engines“, Asia and South Pacific Design Automation Conference, ASP-DAC 2000, Yokohama, Japan, January 2000.
13. R. Kress et al.: „A Datapath Synthesis System for the Reconfigurable Datapath Architecture“; Asia and South Pacific Design Automation Conference, ASP-DAC'95, Makuhari, Chiba, Japan, August 29 - September 1, 1995.
14. R. Hartenstein: “Reconfigurable Computing”; HighSys '99, Sindelfingen, Germany, Oct. 1999
15. C. A. Moritz, D. Yeung, A. Agarwal: „Exploring Optimal Cost-Performance Designs for Raw Microprocessors“; Int'l symposium on FPGAs for Custom Computing Machines, FCCM'98, Napa, CA, April 1998.
16. L. Guerra, M. Potkonjak and J. Rabaey: „A Methodology for Guided Behavioral-Level Optimization“; Proceedings of the 35th Design Automation Conference 1998 (DAC'98), June 15-19, 1998, San Francisco, CA, USA.
17. L. Guerra, M. Potkonjak and J. Rabaey: „System-Level Design Guidance Using Algorithm Properties“; IEEE VLSI Signal Processing Workshop, 1994.
18. W. Pedrycz: „Fuzzy Modelling - Paradigms and Practice“; Kluwer Academic Publishers, 1996.
19. B.R. Gaines: „Foundations of Fuzzy Reasoning“; Int'l Journal of Man-Machine Studies, Vol. 8, 1976.