

# Mapping Applications onto reconfigurable KressArrays

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

Computer Structures Group, Informatik  
University of Kaiserslautern  
D-67653 Kaiserslautern, Germany  
Fax: +49 631 205 2640  
abakus@informatik.uni-kl.de  
<http://xputers.informatik.uni-kl.de>

**Abstract.** This paper introduces a design space explorer for coarse-grained reconfigurable KressArray architectures - to enable the designer to find out the optimal KressArray architecture for a given application. This tool employs a mapper based on simulated annealing, and is highly configurable for a variety of different KressArray architectures. Using performance estimation and other statistic data, the user can interactively change the architecture, until it suits the requirements of the application.

## 1. Introduction

The research area of reconfigurable computing has experienced a rapid expansion in the last few years. In many application areas reconfigurable hardware systems have proven to achieve substantial speed-up over classical (micro)processor-based solutions [1]. The range of these application areas has been extending continuously, including image processing as well as DSP, encryption, pattern recognition and many more.

The implementation of such applications has been done mostly on systems based on fine-grained FPGAs, as they are widely available. However, it turns out that these devices do not suit well for computational applications due to several reasons [2] [3]. As an alternative to fine-grained FPGAs, several coarse-grained reconfigurable architectures have been developed [4][5][6][7][8], which try to overcome the above problems by providing multiple-bit wide datapaths and more complex operators in the processing elements.

An example for such architectures is the KressArray [7], which features arithmetic and logic operators on the level of the C programming language, making the mapping of applications much more simple than for FPGAs. For the first KressArray prototype, which is also known as rDPA (reconfigurable Datapath Architecture), a Datapath Synthesis System (DPSS) [9] based on simulated annealing has been implemented, which accepts input at a high level language. However, after experiments with different KressArray architectures, it has shown, that the available communication resources have a major impact on the efficiency of the mapping. Thus, a KressArray design space

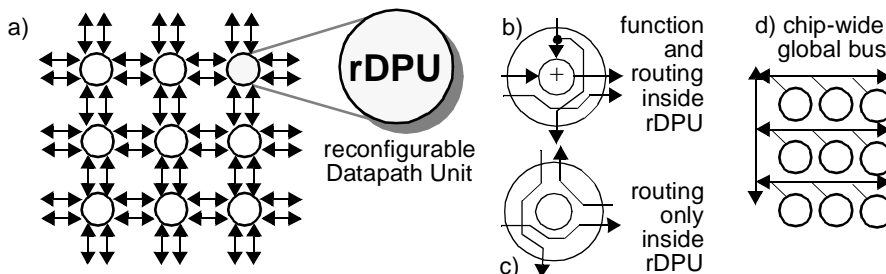


Figure 1: KressArray architecture: a) KressArray-III; b,c) routing inside rDPU; d) global bus

explorer has been implemented, which assists a designer to find the best KressArray architecture for a given problem. This paper concentrates on the simulated-annealing based mapper, which does a placement and routing of a given datapath onto a KressArray architecture. The mapper is highly parametrized in terms of available communication and function resources, which allows experimenting within a short time with a wide variety of different architectures for a given application, or, an entire application domain.

Section 2 briefly summarizes the KressArray having been published elsewhere [7][9][10]. In section 3 the KressArray design space explorer will be introduced. In section 4 the configurable mapper tool is described briefly. The operation of this tool is demonstrated by 2 versions of an example in section 5. Finally, conclusions are drawn.

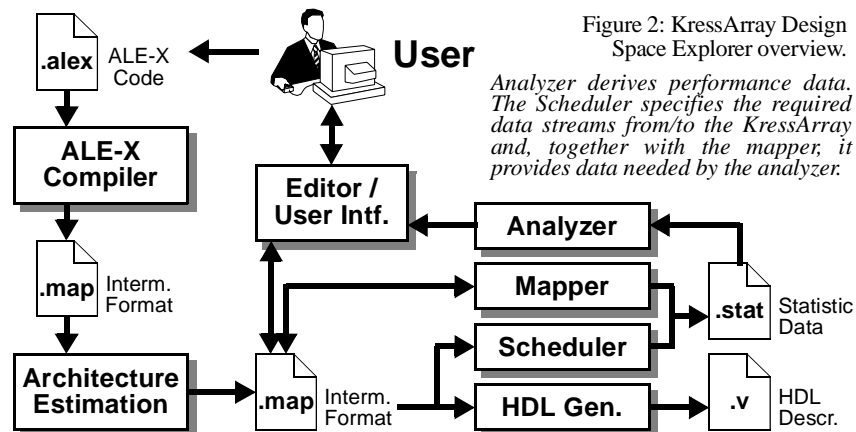
## 2. The KressArray

As an example of the KressArray architecture family [7][9][10], the current new prototype KressArray-III is illustrated in Figure 1a. It consists of a mesh of Processing Elements (PE), also called rDPUs (reconfigurable Datapath Units), which are connected to their four nearest neighbours by two bidirectional links with a datapath width of up to 32 bit, where “bidirectional” means that a direction is selected at configuration time, i. e. is fixed at run time. Nearest neighbour connections (NN links) are used to transfer operands to/from a rDPU, and, to route other data through a rDPU (Figure 1b), as well as for routing through only (Figure 1 c). Besides the NN links, a background communication architecture (called *back buses*) with one global bus (Figure 1 d) or multiple global buses and/or bus segments, i. e. semi-global buses (Figure 3 a and b) provides additional communication resources.

Currently the functionality of each PE consists of the integer operators provided by the programming language C. The mapper, however, also supports rDPUs with other operator repertoires, such as e. g. specialized for accelerator usage in a particular application area. Execution inside PEs is transport-triggered, i. e. it starts as soon as all operands needed are available. The data to be processed and the results to be stored back can be transferred to and from the array in two different ways: over the global bus and over rDPUs ports at the edges of the array.

## 3. Design Space Exploration

An overview of the KressArray design space explorer is given in figure 2. The user provides a high-level description of the application using the high level ALE-X language [9]. The ALE-X compiler generates an expression graph, written in an intermedi-



ate format. This format also describes the properties of the rDPU operator repertory and the KressArray architecture, as well as the mapping added later on (explained later). In a next step, the expression tree is analysed and minimal requirements to the architecture are estimated. The intermediate file is then enhanced by these architecture definitions. In the following design process, the application is mapped onto the KressArray by the mapper tool, which generates a file in the same intermediate format as its input, allowing several consecutive mapping steps. A data scheduler determines an optimized array I/O sequence as well as a performance estimation. Both, mapper and scheduler, generate statistical data like usage of the communication resources and critical path information. From this data, an analyser derives possible enhancements to the architecture, which are presented to the user by an interactive editor. This editor is also used to control the design process itself. When a suitable architecture has been found, a HDL description (currently Verilog) can be generated from the mapping for simulation.

#### 4. The Mapper

The mapper tool maps the application datapath onto the KressArray by placement and routing. The mapper can be seen as an extended version of the KressArray-I mapper [9], which now can handle different KressArray architectures. The mapping algorithm is based on simulated annealing, including a router for the nearest neighbour connections. For the mapper, different architectural properties may be specified:

- The size of the array.
- Areas with different rDPU functionality. For example, in figure 3c, the rDPUs in every second column of the array have a different function set available. The mapper has to consider this when placing the operators.
- The available repertory of nearest neighbour connections (see figure 1). Per side of the cell can be one or multiple unidirectional and/or bidirectional connections.
- One or multiple row and column buses (see figure 3a,b). These buses connect several rDPUs in a row or a column. The buses may be segmented (figure 3b).
- The length of routing paths for nearest neighbour connections.
- The number of routing channels through a rDPU (see figure 1b,c).
- Peripheral port location constraints to support particular communication architectures connecting the array to surrounding memory and other circuitry
- particular placements or groups (e. g. library items) can be frozen (modular mapping)

In addition to the structure of the array, parameters to control the annealing process may be specified, such as e. g. the starting temperature, the number of iterations, and the end temperature. Also, for each of these communication resources, a cost factor for a connection using this resource can be specified. As the mapper produces as output the same intermediate file format as the input, several annealing steps may be performed sequentially, e.g. a low-temperature simulated annealing after a normal mapping.

For each communication resource, as well as for the global bus, the costs for a connection can be specified, which together make up the cost function for the annealing.

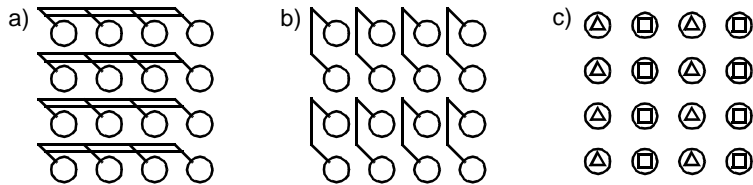


Figure 3: Architectural properties of KressArrays: a) example row buses; b) example column buses (segmented); c) different function sets in alternating columns

This way, specific resources can also be discouraged. Normally, one would assign the global bus a high cost factor, since those connections are slow per se, and, in contrast to routing through cells, by not supporting pipelining. If row or column buses are available, they would normally get a medium cost level, while nearest neighbour connections have the lowest cost, since being preferred. With this strategy, the mapper will eventually find a mapping, which does not use expensive interconnect. Such connect resources can then be removed for the next iteration step in the design space exploration process, leading to a more effective implementation of the KressArray.

## 5. Mapping Applications onto the KressArray

For reconfigurable computing, the use of fine grained FPGAs implies a large connectivity overhead because even each simple operation on dataword widths of 16 or 32 bit massively needs interconnect resources to be implemented on single bit PEs. In spite of approaches like carry chains in contemporary FPGAs[11], a coarse grained architecture involves a much smaller routing problem. That's why we have used a heuristic algorithms, which would require unacceptable computation times for fine grained FPGA mappings. For the KressArray, a simulated annealing approach has been chosen, which gives quite encouraging mapping results. Next subsection shows the efficiency of this approach by an example: mapping the datapath of an image processing filter.

### 5.1 A complex example: SNN Filter

The Symmetric Nearest Neighbour (SNN)-Filter is used as sharpening filter in object detection. In the following, a short introduction to this algorithm is given. The basic operation of this filter is shown in fig. 4. The algorithm calculates a pixel of the result image by considering a 3-by-3 neighbourhood of the original image. First, four neighbouring pixels are selected according to the illustration in fig. 4. The selections are based on the colour distance  $\text{dist}(a,b)$ , which is calculated like shown at the bottom of the source code in the figure, involving colour separation of the pixels. After the selections, the resulting pixel is the average colour value of the four selected pixels.

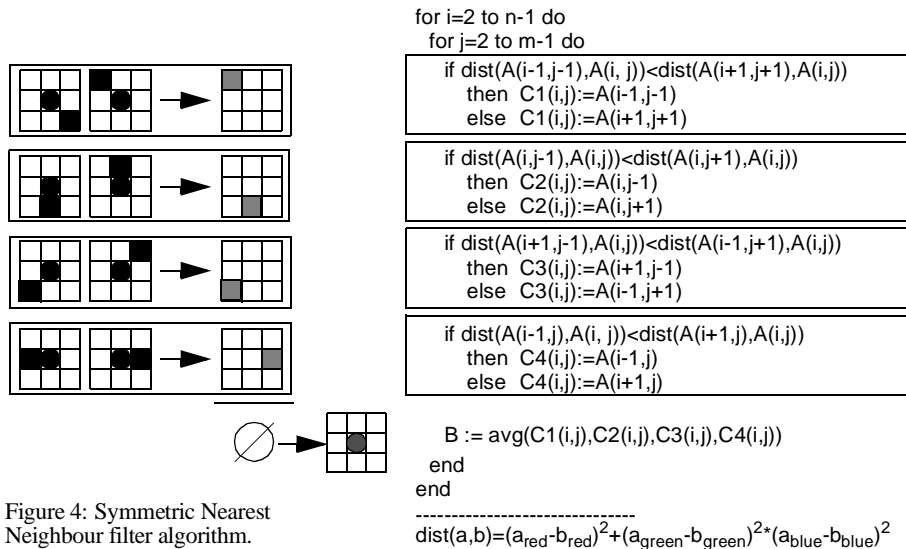


Figure 4: Symmetric Nearest Neighbour filter algorithm.

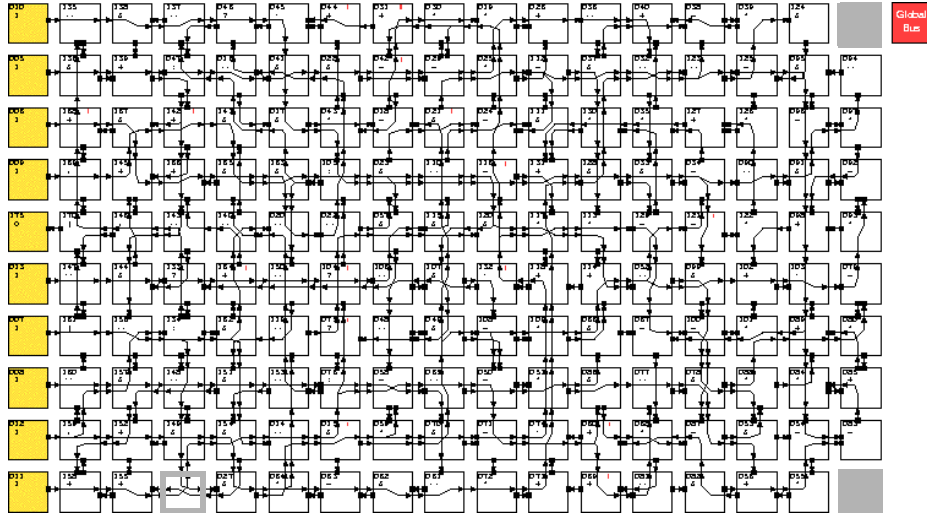


Figure 5: Mapping of a SNN filter on PEs with 8 NN links: 16 back buses (not shown, legend see figure 7).

The result of the first mapping is shown in fig. 5. The algorithm uses 157 PEs and has been mapped onto a 10-by-16 KressArray. In the chosen architecture each PE provides two bidirectional nearest neighbour connections to each of its four neighbour-PEs.

There are nine input ports and one output port, which were all put at the western side of the array, leaving the exact placement up to the annealer again (the output port is the fifth one from top). As the nearest-neighbour resources are used up in some places, additional global bus connections are necessary. These can be identified by the small bars in the upper right corner of the PEs. The mapping in fig. 5 uses 16 bus connections, which is a moderate value for a datapath of such a complexity. However, for the I/O, no global bus connections were needed. Also, the neighbour connections have a quite good utilization, which is also caused by the fact, that in this algorithm many output values of PEs are used multiple times. E. g. eight of the nine input values are used at four different PE inputs. The mapping took about 24 minutes on a Pentium-II 366-MHz PC.

The design space explorer enables the designer to map his algorithm on different architectures and to compare the resulting structures. Depending on the chosen KressArray architecture the mappings will differ in many points like the number of used PEs, the used communication resources, etc. With this information the designer can decide his compromise between complexity (and costs) of the base-architecture and the execution time.

In the example above an alternative architecture could be the same KressArray (10-by-16) with three nearest-neighbour connections (*NN links*) in each direction instead of two. Then the number of required PEs is still 157 because the number of operations remains the same and the operators provided by the PEs are also unchanged. But, due to the enhanced routing capabilities of the PEs, some connections which were realized by the global background bus (*back bus*) in the first mapping can now be routed using NN links. This explains why in the second mapping only seven bus connections are needed.

The SNN example raises an important question regarding the granularity of the configurable architecture. For the example mappings, the datapath description was fed directly in the DPSS system. However, the direct mapping is not always optimal. E.g. for the colour separation, two PEs were used, one for a shift operation and the next for

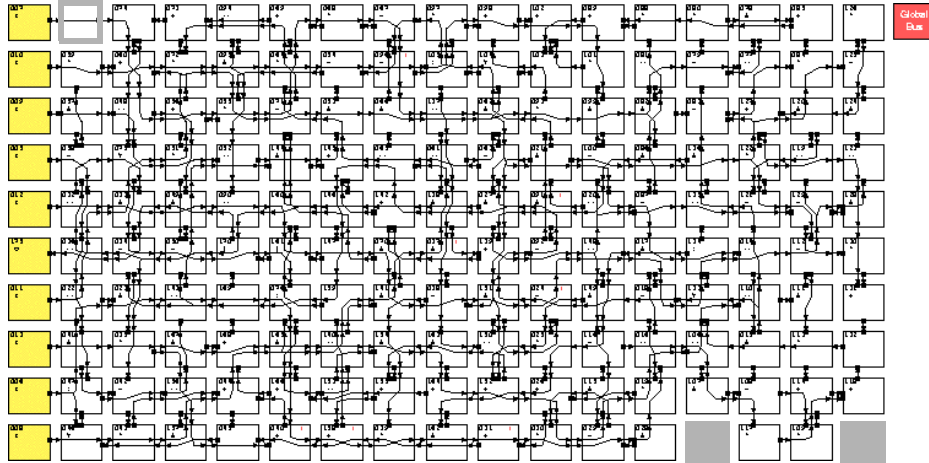


Figure 6: Mapping of a SNN filter on PEs with 12 NN links: only 7 back buses needed (legend: figure 7).

masking out the according value. It would not increase the complexity of the PEs to provide a function, which combines these two operations, resulting in less PE usage.

## 6. Conclusions

We have introduced a design space exploration environment for the KressArray architecture family. The KressArray is a coarse-grained reconfigurable architecture, which allows much simpler application mapping than fine-grained FPGAs. A highly flexible configurable mapper has been introduced, which is capable of handling a wide variety of different architectures. Based on the mapper, the design space explorer allows the user to find the best suitable KressArray for a given application domain.

- rDPU not used
- used for routing only
- operator and routing
- port location marker

Figure 7: Legend.

## Literature

1. W. Mangione-Smith, et. al.: Seeking Solutions in Configurable Computing; IEEE Computer, Dec. 1997.
2. R. Hartenstein (invited paper): The Microprocessor is no longer General Purpose: why Future Reconfigurable Platforms will win; ISIS'97, Austin, Texas, U.S.A., Oct. 1997
3. R. Hartenstein (opening keynote): Next Generation Configware merging Prototype and Product; RSP'99, Int'l Workshop on Rapid Prototyping, Leuven, Belgium, June 3 - 5, 1998
4. 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.
5. E. Waingold et al.: „Baring it all to Software: Raw Machines“, IEEE Computer 30, pp. 86-93.
6. C. Ebeling, D. Cronquist, P. Franklin: „RaPiD: Reconfigurable Pipelined Datapath“, Workshop on Field Programmable Logic and Applications, FPL'96, Darmstadt, Germany, 1996.
7. R. Kress: „A Fast Reconfigurable ALUs for Xputers“, Ph.D. thesis, Univ. Kaiserslautern, 1996.
8. 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
9. R. Kress et al.: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; ASP-DAC'95, Makuhari, Chiba, Japan, Aug. 29 - Sept. 1, 1995.
10. J. Becker et al.: Parallelization in Co-Compilation for Configurable Accelerators; Asian South Pacific Design Automation Conference 1998 (ASP-DAC'98), Yokohama, Japan
11. N. N.: The Programmable Logic Data Book, Xilinx Inc., San Jose, California, 1998