

# Hardware/Software Co-Design for data-driven Xputer-based Accelerators

Reiner W. Hartenstein, Jürgen Becker

University of Kaiserslautern

Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany

Fax: ++49 631 205 2640, e-mail: abakus@informatik.uni-kl.de

URL: <http://xputers.informatik.uni-kl.de/>

## Abstract

*The paper presents the parallelizing programming environment CoDe-X introducing hardware/software co-design strategies on two levels of partitioning for data-driven Xputer-based accelerators. CoDe-X performs both, in the first level a profiling-driven host/accelerator partitioning for performance optimization, and in a second level a resource-driven sequential/structural partitioning of the accelerator source code to optimize the utilization of its reconfigurable resources. CoDe-X accepts a C dialect also including optional data-procedural language features, which can be included to achieve highest possible acceleration factors provided by the Xputer hardware.*

## 1. Introduction

Today, increasingly complex and computation-intensive tasks have to be performed by computers. From empirical studies [30] it can be concluded that the major amount in computation time is due to rather simple loop constructs. Since additionally these loop constructs are combined with indexed array data structures, ordinary von Neumann style computers are burdened mainly with addressing computations. First efforts to reduce addressing overhead and to introduce parallelism have been undertaken by the development of pipelined and vector supercomputers [23], [17]. Together with the achievements in supercomputer technology, parallelizing compilers have been developed, where compilation is based on data dependence analysis [2], [32] producing executable code for different parallel target machines. Unfortunately, the hardware structures are not reflecting the structure of the algorithms very well, which restricts the exploitation of inherent parallelism in the algorithm to a large extent.

Emanating from the technology of *field-programmable logic* (FPL) the new paradigm of structural programming has evolved. Instead of loading the program code sequentially into memory (*procedural programming*), hardware structures are configured to fulfill the application needs (*structural programming*). With the development of Custom Computing Machines (CCMs: [13], [16], [24]) the combination of structural and procedural programming has been evolved by FCCMs [4] [5]. The scene of hardware/software co-design has introduced a number of approaches to speed-up performance, to optimize hardware/software trade-off and to reduce total design time [6], [7], [21]. The von Neumann paradigm (used in

the FCCM-approach) does not efficiently support “soft” hardware because of its extremely tight coupling between instruction sequencer and ALU: architectures fall apart, as soon as the data path is changed. So a new paradigm is desirable like the one of Xputers, which conveniently supports “soft” ALUs like the rALU concept (reconfigurable ALU) [11] or the rDPA (reconfigurable data path array) [18] [19].

In such an environment parallelizing compilers require two levels of partitioning: host/accelerator partitioning for optimizing performance and a structural/sequential partitioning (second level) for optimizing the hardware/software trade-off of the Xputer resources. Furthermore the application development environment CoDe-X combines three programming paradigms into one more powerful approach: the control procedural paradigm reflected in C language features, the data procedural paradigm realized in an optional language extension for specifying selected data procedural application parts executed faster by Xputer hardware and the structural programming paradigm for the reconfigurable Xputer hardware components .

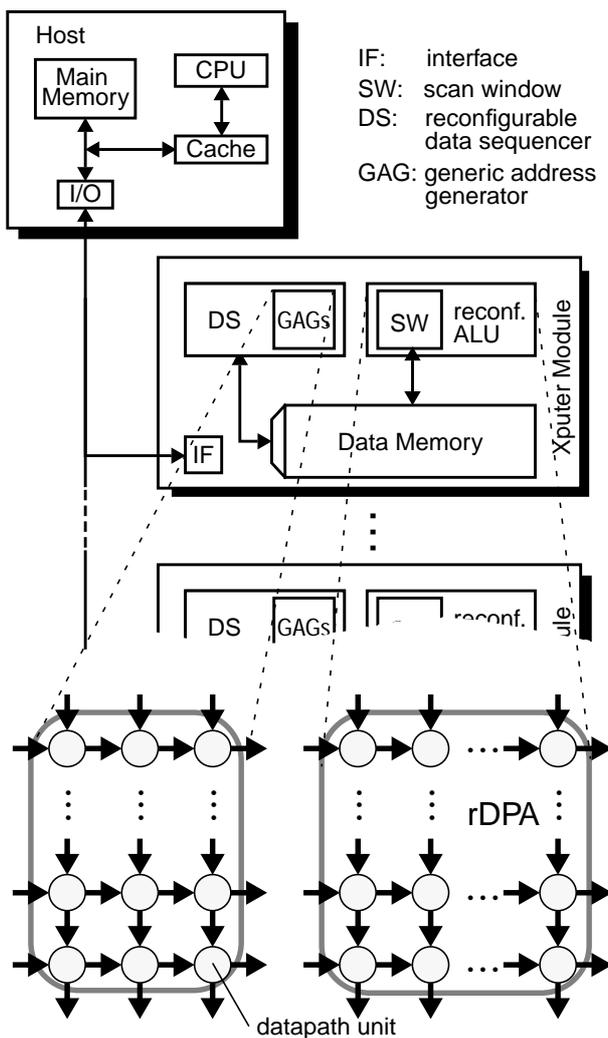
To stress the significance of this application development methodology, the paper first gives an introduction to the underlying hardware platform. Section 3 presents the co-design framework CoDe-X and its strategies including the option of additional data procedural features for experienced users. In section 4 computation-intensive application examples from the area of image processing are discussed.

## 2. Xputer-based Accelerators

Many applications require the same data manipulations to be performed on a large amount of data, e.g. statement blocks in nested loops. The Xputer machine paradigm aims at the acceleration of such applications. Xputers are especially designed to reduce the von Neumann bottleneck of repetitive decoding and interpreting address and data computations. High performance improvements have been achieved for the class of regular, scientific computations [3], [10], [11], [12].

Xputer-based accelerators consist of several (up to seven) Xputer modules. The modules are connected to a host computer. Making use of the host simplifies disk access and all other I/O operations. After setup, each Xputer module runs independently from the others and the host computer until a complete task is processed. Each module generates an interrupt to the host when the task is finished. So, the host is free to





**Figure 1. Host with several Xputer modules**

concurrently execute other tasks in-between. This allows the use of the Xputer modules as a general purpose acceleration board for time critical parts in an application. The basic structure of an Xputer module consists of three major parts (figure 1):

- the data memory
- the reconfigurable arithmetic & logic unit (rALU) including several rALU subnets with multiple scan windows (SW)
- the reconfigurable data sequencer (DS) comprising several generic address generators (GAGs)

The data memory which is local on each Xputer module, is primarily organized two-dimensionally, but can also be interpreted as higher dimensional. The data is arranged in a special order to optimize the data access sequences, called data map. The scan windows serve as interface of the rALU subnets to the data memory. The main memory of the host and the local memories are logically a single shared memory. A large amount of input data is typically organized in arrays where the array elements are referenced as operands of a computation in a current iteration. The sequence of data accesses in iterations

shows a regularity which allows to describe this sequence by a number of parameters. The reconfigurable generic address generators (GAGs) of the data sequencer (DS) interpret these parameters and compute generic address sequences to access the data. This results in a controlled movement of the scan windows over the data map, each controlled by a single GAG. Each time the scan windows move one step, a compound operator of the corresponding rALU is evaluated. Thus the data sequencer represents the main control instance of an Xputer. The implementation of the data sequencer can be done with reconfigurable logic. The control part is admitted with features for reconfiguration and realized with a single Xilinx XC4013 FPGA. Because of the wide datapath of the generic address generator such a fine grained structure is not suitable. Therefore the reconfigurable datapath architecture (rDPA) is used [8]. In the rDPA, complete arithmetic and logic operators up to 32 bit can be implemented into a single logic block which is called datapath unit (DPU) [18]. In the current prototype the rDPA serves as rALU and consists of 72 identical word-oriented DPUs, which are required for evaluation of operations in standard C programs. For further information about the rDPA approach and its synthesizing software as well as the Xputer hardware see [15], [18], [19].

All I/O operations are done by the host as well as the memory management by taking advantage of the functionality of the host's operating system. The host has direct access to all memories on the Xputer modules via a small bus interface (IF, see figure 1). On the other hand the Xputer can access data in the host's memory. The Xputer activation, its synchronization and data transfers with the host are controlled by the Xputer Run Time System (XRTS) [25].

### 3. The H/S Co-Design Framework CoDe-X

For above described hardware platform, a partitioning and parallelizing compilation framework, CoDe-X, is being implemented. CoDe-X is based on two-level hardware/software co-design strategies and accepts X-C source programs (figure 2). X-C (Xputer-C) is a C dialect. CoDe-X consists of a 1<sup>st</sup> level partitioner (partially implemented), a GNU C compiler, and an X-C compiler (fully implemented). The X-C source input is partitioned in a first level into a part for execution on the host (host tasks, also permitting dynamic structures and operating system calls) and a part for execution on the Xputer (Xputer tasks). Parts for Xputer execution are expressed in a C subset, which lacks only dynamic structures [27]. At second level this input is partitioned by the X-C subset compiler in a sequential part for the DS, and a structural part for the rDPA. Additionally experienced users may also include directly data procedural MoPL-code (Map-oriented Programming Language [1]) into the original C description of applications, or can add new functions described in MoPL to the library such that each user can access them (see figure 2). Less experienced users may use these special MoPL library function similar to C function calls to take full advantage of the high acceleration factors possible by the Xputer paradigm. In this chapter the profiling-driven host/Xputer partitioning is sketched briefly. Then the inclusion of data procedural options for experienced and less experienced users is explained.



### 3.1 Profiling-driven Host/Accelerator Partitioning

The first level of the partitioning process is responsible for the decision of allocating tasks on the Xputer or on the host. Generally three kinds of tasks can be determined:

- host tasks which contain dynamic structures
- Xputer tasks, and
- Xputer-library functions.

The host tasks have to be evaluated on the host, since it cannot be performed on the Xputer accelerator. The Xputer tasks are the candidates for the performance analysis on the host and on the Xputer. For host performance evaluation here similar techniques like Malik and others are used [22], [26]. For Xputer performance analysis generated files from the resource-driven second partitioning level are used (see section 3.3, [18] and [27]). The Xputer-library functions are used to get the highest performance out of the Xputer. An experienced user has developed a function library with all Xputer-library functions together with their performance values. These functions can be called directly in the input C-programs. They are evaluated in any case on the Xputer.

The Xputer tasks are the candidates for the partitioning process in the first level. The user may advise the partitioner about parts to be Xputer-executed in any case. For library functions a list of their performance data is available. First, program parts using MoPL extensions or containing loops which can be transformed by the X-C compiler, are routed for Xputer execution. Such an Xputer job is divided into the following Xputer tasks computable on a single Xputer module:

- basic block (a linear code sequence),
- basic block including conditions,
- fully nested loops (up to seven loops), or

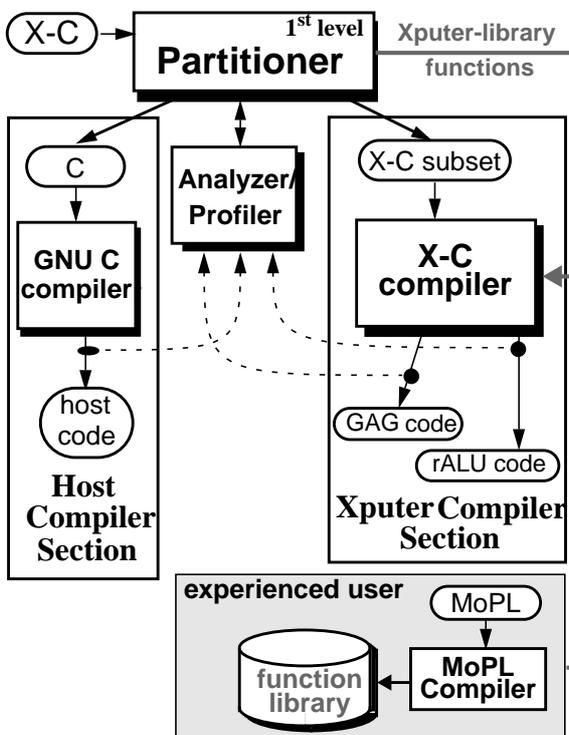


Figure 2. Overview on the CoDe-X Framework

- not fully nested loops (up to seven loops).

A number of possible tasks are generated where basic blocks may be in several tasks. The resulting task graph has several alternative paths in such a case. The restriction to seven nested loops is due to limited resources in the current Xputer prototype. The data sequencer of this prototype contains seven GAGs, which may run in parallel. Here optimization techniques [20] like strip mining (transforming original loop into chunks of approximately equal size for parallel execution), loop distribution (splitting of one loop into two loops computing the same), like loop fusion (transforming two adjacent loops into a single loop) and like loop interchanging (switching inner and outer loop) are also performed by the 1<sup>st</sup> level partitioner in order to exploit potential parallelism and to optimize utilization of the Xputer hardware resources. Different parallel task allocations are analyzed in an iterative partitioning process based on simulated annealing [29] in order to optimize the overall application execution time. Here also the synchronization, communication and reconfiguration overhead during run-time is considered. For more details about the profiling-driven first level partitioning see [8], [9], [14].

### 3.2 Optional Data-Procedural Features

The CoDe-X framework contains an optional data-procedural extension, where experienced users can use and build a generic function library with a large repertory of Xputer-library functions. These should be functions, which are well suited for Xputer use resulting in high acceleration factors compared to single processor workstations e. g. different kind of filters, matrix operations etc. The scan patterns as well as the rALU compound operators can be described in the Xputer language MoPL [1] fully supporting all hardware features.

The input specification is then compiled into Xputer code, namely the datamap, the data sequencer code and the rALU code, dependent on the concrete functions calls in the input X-C-program, which are similar to C-function use. A difference is the use of generic parameters, where the dimensions of actual used parameters have to be specified (see figure 3). In this example a 8 by 10 matrix is multiplied

```
#include <stdio.h>
#include <MoPL.h>
int x[10];
int b[10];
int A[10][10];

int main
{
    /* Xputer-library function call for multiplying a
    8x10-matrix with a 10-component vector */
    MatrixVekMul(x,A,b,8,10,10);
}
```

Figure 3. Example of including a data procedural function call into an X-C program



with a 10-component vector. The parameter list defines the interface to the rest of the program. One Xputer-library function call corresponds to one Xputer task in the first level partitioning process and is executed in every case on an Xputer-based accelerator. Here the user has the possibility to influence the partitioning process. So less experienced users can only apply Xputer-library functions available in standard libraries, which are included to the X-C program like normal C libraries (figure 3). There is no need to build new library functions described in MoPL, if a user is not familiar doing this. But if one experienced wants to create new Xputer-library functions, he can describe the required functions in MoPL (described in detail including examples in [1]) and include them in an existing library or create a new library. As X-C is an ANSI C extension, it is also possible to program directly MoPL-parts into C-programs (figure 3), if no suitable library function is available. Optional data-procedural language features make highest acceleration factors provided by Xputer hardware possible for users familiar with MoPL. An example for using Xputer-library functions as well as including directly MoPL code into an X-C program is given in section 4.

### 3.3 Resource-driven second level Partitioning

The X-C compiler realizes the 2<sup>nd</sup> level of partitioning and translates an X-C program into code which can be executed on the Xputer. The compiler performs a data and control flow analysis. First the control flow of the program graph is partitioned according to the algorithm of Tarjan [31] resulting in a partial execution order. This algorithm is partitioning the program graph into subgraphs, which are called *Strongly Connected Components*. These components correspond to connected statement sequences like fully nested loops for example, which are possibly parallelizable. The Xputer hardware resources are exploited in an optimized way by analyzing, if the structure of statement sequences can be mapped well to the Xputer hardware providing instruction level parallelism and avoiding reconfigurations or idle hardware resources [27]. Further details and examples about the X-C compiler can be found in [15], [27], [28].

### 4. Example: Image Processing Algorithm

Smoothing-operations are used primarily for diminishing spurious effects, that may be present in a digital image as a result of a poor sampling system or transmission channel. Neighborhood averaging is a straightforward spatial-domain technique for image smoothing. Given an N x N image  $f(x,y)$ , the procedure is to generate a smoothed image  $g(x,y)$ , whose gray level at each point  $(x,y)$  is obtained by averaging the gray-level values of the pixels of  $f$  contained in a predefined neighborhood (kernel) of  $(x,y)$ . In other words, the smoothed image is obtained by using the equation:

$$g(x, y) = \frac{1}{M} \sum_{(n, m) \in S} f(n, m) \quad (\text{eq. 1})$$

for  $x,y = 0,1, \dots, N-1$ .  $S$  is the set of coordinates of points in the neighborhood of (but not including) the point  $(x,y)$ , and  $M$  is a pre-computed normalization factor. This small example for

illustrating the methods of CoDe-X was divided in four tasks. Two of them were filtered out in the first step for being executed on the host in any case. These were tasks containing I/O routines for reading input parameters and routines for plotting the image, which cannot be executed on the Xputer. The remaining two tasks were potential candidates for mapping onto the Xputer. In one of these two tasks strip mining was applied by the 1<sup>st</sup> level partitioner. The resulting smaller independent loops can be executed in parallel on different Xputer modules. The X-C compiler was performing loop unrolling additionally up to the limit of available hardware resources (see section 3.3). The final acceleration factor in comparison with a SUN SPARC 10/51 was 73. For details about this example see [14].

Another application from the area of image processing are sharpening algorithms, which are special cases of the discrete convolution for detecting edges. In figure 4 a data procedural function named "SharpImage" is called within an X-C program performing a sharpening over a 640 by 400 pixel image by using a 3 by 3 coefficient matrix. This Xputer library

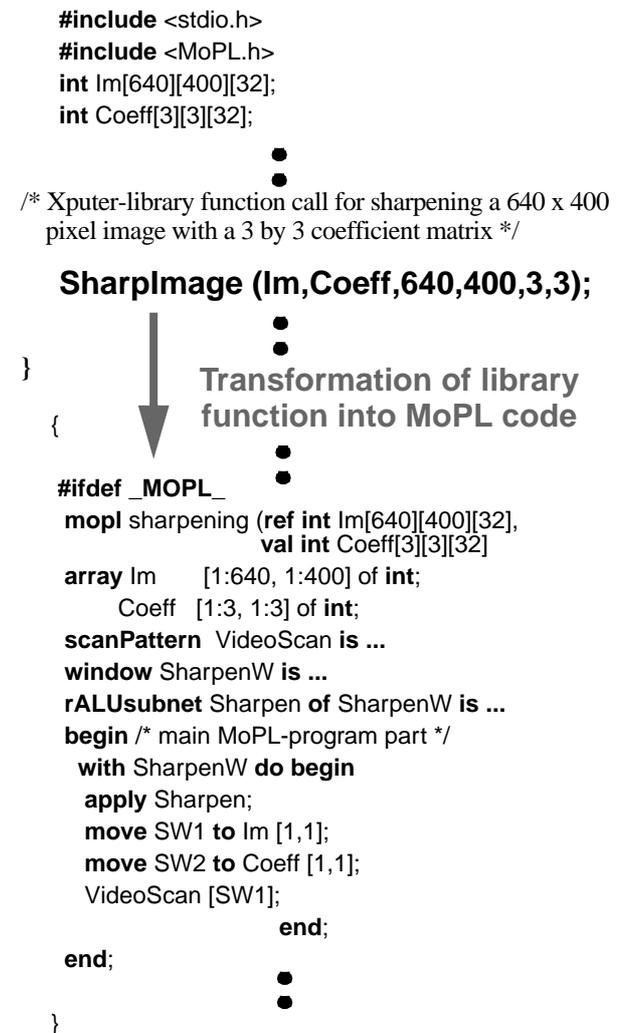


Figure 4. Transformation of a data procedural function into corresponding MoPL-code



function will be executed in any case by the accelerator and is transformed by CoDe-X into the corresponding MoPL-code optimized for Xputer hardware execution. First the parameter transfer is done, then different declarations (*scan pattern*, *window*, *rALUsubnet*) are performed and finally the main program part applies a video scan over the image computing in every step the sharpening-value of one pixel. In figure 4 the MoPL-code is outlined only briefly, for details about programming with MoPL please see [1]. If no library function for sharpening would have been available, the MoPL-code in figure 4 could also be included directly into the X-C program by experienced users.

## 5. Conclusions

A partitioning and parallelizing programming environment for Xputers has been presented. The Xputer is used as universal accelerator hardware based on reconfigurable datapaths. Before code generation CoDe-X carries out an automatic two level optimizing partitioning for profiling-driven host/accelerator partitioning (1st level) and resource-parameter-driven procedural/structural partitioning (2nd level) for accelerator programming. From a number of application examples encouraging acceleration factors have been obtained in comparison to workstation-only versions [14]. A novel characteristic of CoDe-X is the combination of the control and data procedural paradigm into one input specification language X-C, whereas the structural programming paradigm for the reconfigurable Xputer hardware components is transparently for users integrated into the synthesis path of CoDe-X. Thus three different programming paradigms are combined into one more powerful compilation approach.

## 6. References

- [1] A. Ast, J. Becker, R. W. Hartenstein, R. Kress, H. Reinig, K. Schmidt: Data-procedural Languages for FPL-based Machines; 4th Int. Workshop on Field Programmable Logic and Appl., FPL'94, Prague, Sept. 7-10, 1994, Lecture Notes in Computer Science, Springer, 1994
- [2] U. Banerjee: Dependence Analysis for Supercomputing; Kluwer, 1988.
- [3] Jürgen Becker, Reiner W. Hartenstein, Rainer Kress, Helmut Reinig: High-Performance Computing Using a Reconfigurable Accelerator; Proc. of Workshop on High Performance Computing, Montreal, Canada, July 1995
- [4] D. Buell, K. L. Pocek: Proc. IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, Napa, CA, April 1993
- [5] see [4], but 1994, 1995, and 1996
- [6] R. Ernst, J. Henkel, T. Benner: Hardware-Software Cosynthesis for Microcontrollers; IEEE Design & Test, pp. 64-75, Dec. 1993
- [7] R. K. Gupta, C. N. Coelho, G. De Micheli: Program Implementation Schemes for Hardware-Software Systems: IEEE Computer, pp. 48-55, Jan. 1994
- [8] R. Hartenstein, J. Becker, M. Herz, R. Kress, U. Nageldinger: A Parallelizing Programming Environment for Embedded Xputer-based Accelerators; High Performance Computing Symposium '96, Ottawa, Canada, June 1996
- [9] Reiner W. Hartenstein, Jürgen Becker, Rainer Kress, Helmut Reinig: CoDe-X: A Novel Two-Level Hardware/Software Co-Design Framework; Proc. of VLSI Design 96 Conf., Bangalore, India, January 1996
- [10] R. W. Hartenstein, J. Becker, R. Kress, H. Reinig, K. Schmidt: A Reconfigurable Machine for Applications in Image and Video Compression; Conf. on Compression Techniques & Standards for Image & Video Compression, Amsterdam, Netherlands, 1995
- [11] R.W. Hartenstein, A.G. Hirschbiel, M. Weber: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware; InfoJapan'90-International Conference memorizing the 30th Anniversary of the Computer Society of Japan, Tokyo, Japan, 1990;
- [12] see [11]; also in: Future Generation Computer Systems no. 7, pp. 181-198 (North-Holland, 1991/92)
- [13] R. Hartenstein, (opening key note): Custom Computing Machines - An Overview; Workshop on Design Methodologies for Microelectronics, Smolenice Castle, Slovakia, Sept. 1995
- [14] Reiner W. Hartenstein, Jürgen Becker, Rainer Kress: Two-Level Hardware/Software Partitioning Using CoDe-X; Int. IEEE Symp. on Engineering of Computer Based Systems (ECBS), Friedrichshafen, Germany, March 1996
- [15] R. W. Hartenstein, R. Kress: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; ASP-DAC'95, Nippon Convention Center, Makuhari, Chiba, Japan, Aug. 29 - Sept. 1, 1995
- [16] R. Hartenstein, J. Becker, R. Kress: Custom Computing Machines vs. Hardware/Software Co-Design: From a globalized point of view; 6th Int'l. Workshop on Field Programmable Logic and Applications FPL'96, Darmstadt, 1996
- [17] K. Hwang: "Advanced Computer Architecture: Parallelism, Scalability, Programmability"; McGraw-Hill, 1993.
- [18] R. Kress: A fast reconfigurable ALU for Xputers; Ph. D. dissertation, Kaiserslautern University, 1996
- [19] R. Kress: Computing with reconfigurable ALU arrays; IT Press (planned for 1997)
- [20] D. B. Loveman: Program Improvement by Source-to-Source Transformation; Journal of the Association for Computing Machinery, Vol. 24, No. 1, pp.121-145, January 1977
- [21] W. Luk, T. Lu, I. Page: Hardware-Software codesign of Multidimensional Programs; Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, Napa, CA, April 1994
- [22] S. Malik, W. Wolf, A. Wolfe, Y.-T. Li, T.-Y. Yen: Performance Analysis of Embedded Systems; NATO/ASI, Tremezzo, Italy, 1995, Kluwer Acad. Publishers, 1995
- [23] D.I. Moldovan: Parallel Processing - From Applications to Systems; Morgan Kaufmann Publ., 1993.
- [24] D. Monjau: Proc. GI/ITG Workshop Custom Computing, Dagstuhl, Germany, June 1996
- [25] U. Nageldinger: Design and Implementation of a menu-driven Run Time System for the MoM-3; Master Thesis, University of Kaiserslautern, 1995
- [26] P. Puschner, Ch. Koza: Calculating the Maximum Execution Time of Real-Time Programs; Journal of Real-Time Systems p. 159-176, Kluwer Academic Publishers 1989
- [27] K. Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph.D. Thesis, Univ. of Kaiserslautern, 1994
- [28] K. Schmidt: Xputers: a high performance computing paradigm; IT Press (planned for 1997)
- [29] N. A. Sherwani: Algorithms for Physical Design Automation; Kluwer Academic Publishers, Boston 1993
- [30] Z. Shen, Z. Li, P.-C. Yew: An Empirical Study of Fortran Programs for Parallelizing Compiler; IEEE Trans. on Parallel and Distrib. Syst., Vol.1, No.3, pp. 356-364, July 90.
- [31] R. E. Tarjan: Testing Flow Graph Reducibility; Journal of Computer and System Sciences 9, pp. 355-365, 1974
- [32] H.P. Zima, B. Chapman: Supercompilers for Parallel and Vector Computers; ACM Press Frontier Series, Addison-Wesley, 1990

