Reiner W. Hartenstein, Jürgen Becker: A Two-level Co-Design Framework for data-driven Xputer-based Accelerators; Proc. of 30th Anual Hawaii Int. Conf. on System Science (HICSS-30), January 7-10, Wailea, Maui, Hawaii, USA, 1997

A Two-level Co-Design Framework for Xputer-based data-driven reconfigurable Accelerators

Reiner W. Hartenstein, Jürgen Becker
Universitaet Kaiserslautern
Erwin-Schroedinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, e-mail: hartenst@rhrk.uni-kl.de
URL: http://xputers.informatik.uni-kl.de/

Abstract

The paper presents the parallelizing compilation environment CoDe-X for simultaneous programming of Xputer-based accelerators and their host. The paper introduces its hardware/software co-design strategies at two levels of partitioning. CoDe-X performs both, at first level a profiling-driven host/accelerator partitioning for performance optimization, and at second level a resource-driven sequential/structural partitioning of the accelerator source code to optimize the utilization of its reconfigurable resources. To stress the significance of this application development methodology, the paper first gives an introduction to the underlying hardware platform.

1. Software-only Accelerator Implementation

Very often hardware and software are alternatives to implement the same algorithm. Hardware is the *structural implementation*, whereas software results in a *procedural implementation*. We may distinguish two different worlds of computation: computation in space (structural), and, computation in time (procedural). At the area of systolic arrays, also called ASAPs (application-specific array processors), both worlds overlap. ASAP synthesis methods use time and space within the same formula, where mappings are links between both worlds: an emerging dual new computing science.

Hardware has become soft. Emanating from the technology of *field-programmable logic* (FPL) the new paradigm of *structural programming* has evolved. So we now have two programming paradigms: programming in time and programming in space. Now it is time to become aware of the really existing two kinds of software:

- procedural software (code downloaded to RAM)
- structural software (downloaded to hidden RAM)

Tools like XACT and others are the code generators for structural software. The R&D area of Custom Computing Machines (CCMs: [1], [2], [4]), such as FPGA-based CCMs (FCCMs [5] [6]), merge both kinds of software to an integrated methodology to speed-up performance. We have obtained a dual software-only implementation: procedural software running on the host, together with structural software running on the reconfigurable accelerator.

Implementing CCMs has a lot in common with hardware/ software co-design to optimize hardware/software trade-off and to reduce total design time [7], [8], [9]. The main difference is the target platform: the structural *program* is frozen into ASICs or other hardware structures. Instead of *structural programming* we call this *hardware synthesis*.

With the FCCM approach implementations on a von Neumann host are accelerated by using add-on field-programmable circuits like FPGAs, to be *structurally programmed*. But currently this ”structural programming“ usually requires hardware experts, since contemporary FPGA-based accelerator architectures are far from being general purpose platforms. FPGAs currently available commercially have several severe draw-backs:

- by far too area-inefficient
- designed for random logic only

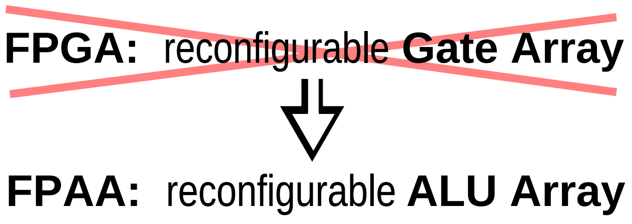

~~FPGA: reconfigurable Gate Array~~
FPAA: reconfigurable ALU Array

Figure 1. Toward field-programmable ALU Arrays.

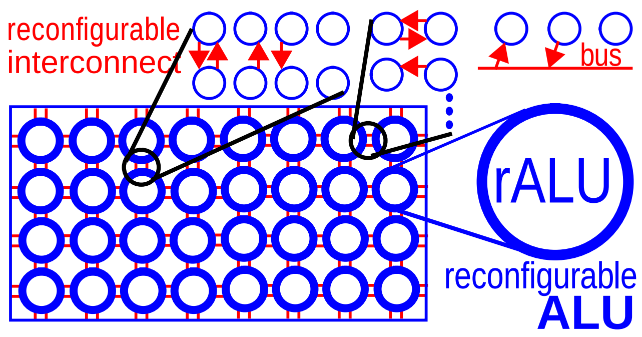


Figure 2. Illustrating the Kress Array Principles.

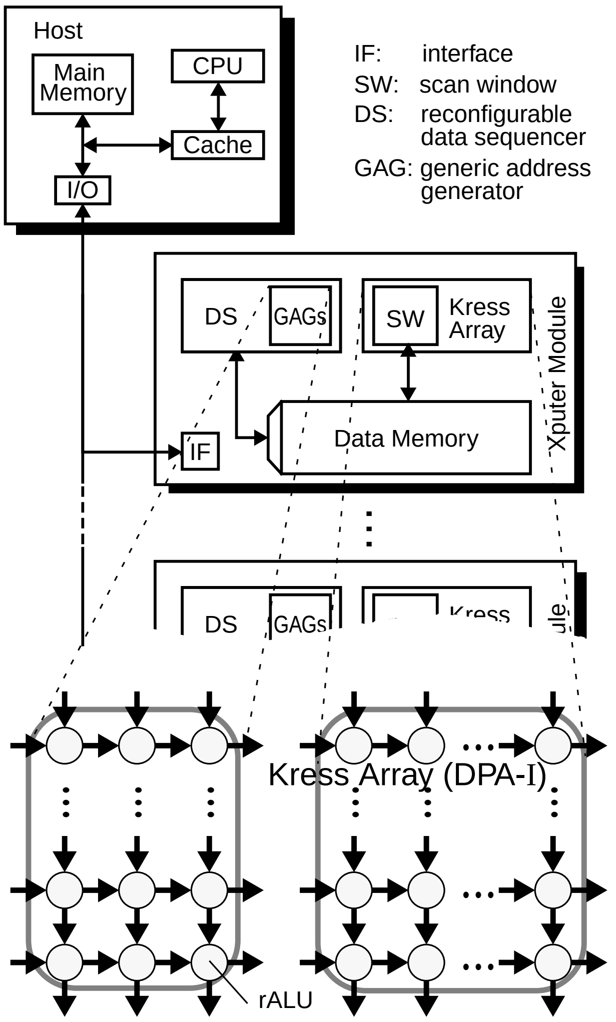


Figure 3. Host with several Xputer modules

- too fine grain and too slow
- poor cost/performance ratio for highly computing-intensive applications

The problem is the wide variety of architectures urged by optimization needs which stem from these draw-backs. Neither in CCMs nor in hardware/software co-design a common model is available. The von Neumann paradigm does not efficiently support ”soft“ hardware because of its tight coupling between instruction sequencer and ALU [10]. You need a new instruction sequencer, as soon as the data path is changed by structural programming: the architecture falls apart.

2. The Kress ALU Array

A new kind of structurally programmable technology platform is needed. But also a new paradigm is needed like the one of Xputers (figure 7), which conveniently supports *soft ALUs* like in the *rALU Array* concept *(reconfigurable ALU Array)* [11]. For more about Xputers see section 3.

To support highly computing-intensive applications we need

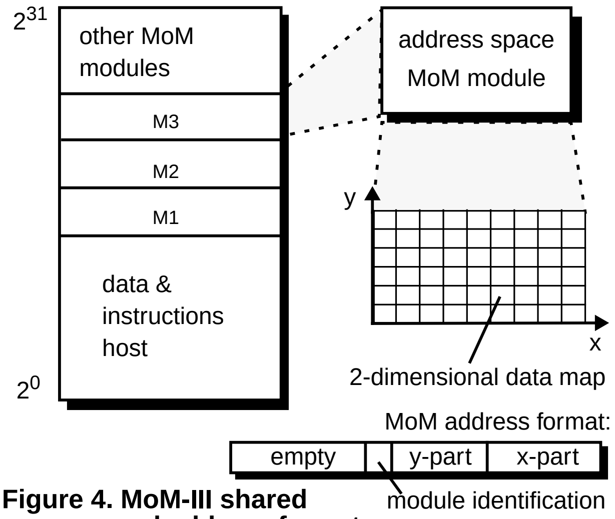


Figure 4. MoM-III shared memory and address format

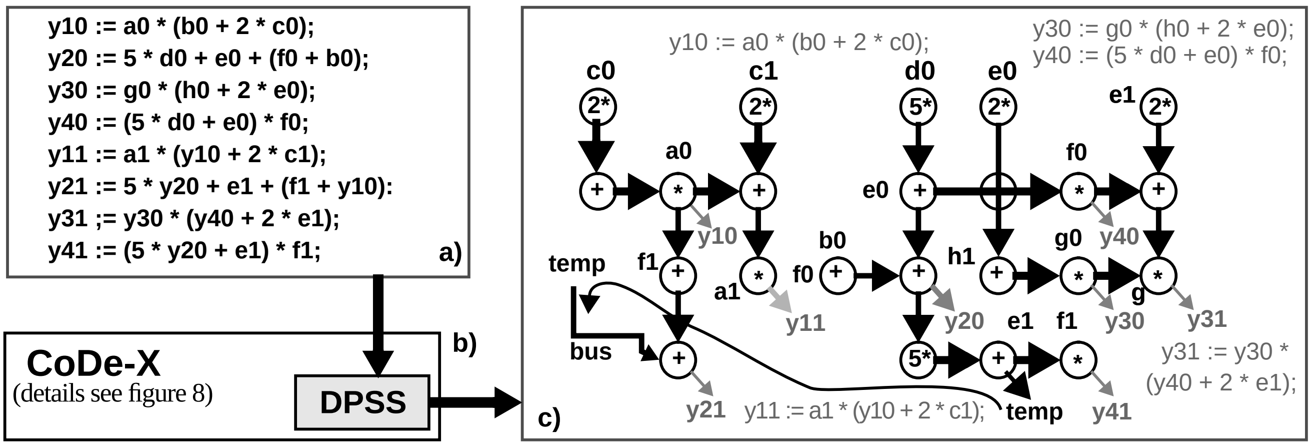
structurally programmable platforms providing word level parallelism instead of the bit level parallelism of FPGAs. We need FPAAs (field-programmable ALU Arrays) instead of FPGAs (figure 1). For area efficiency this new platform should be suitable for full custom design, like known from ASAPs operating in SIMD mode. But ASAPs are not structurally programmable and support only problems with completely regular data dependencies.

What we need is a platform as dense as ASAPs, but structurally programmable. For flexibility to implement applications with highly irregular data dependencies, each PE (processing element) should be programmable individually. Each PE should be a reconfigurable ALU (rALU). Also the local interconnect between particular PEs or rALUs should be programmable individually.

A good solution is the Kress Array, illustrated by figure 2. It permits to map highly irregular applications onto a regularly structured hardware platform. Figure 5 shows a mapping example. Eight equations expressed in C language (figure 5 a) are mapped onto the Kress Array in figure 2 by the DPSS subsystem (figure 5 b). The result of this structural programming effort is the configured data path within the Kress ALU Array (figure 5 c: only internal data paths shown). DPSS (Data Path Synthesis System [12], [13]) is a simulated annealing optimizer, which carries out placement and routing [15]. This example also illustrates part of the role of the CoDe-X application compiler introduced later in this paper. The Kress Array instruction level parallelism has following features.

- transparently scalable
- fast routing and placement (seconds only)
- dynamic partial reconfiguration (microseconds)
- suitable for full custom design
- much higher acceleration than by caches
- fast and low power by full custom design

Reiner W. Hartenstein, Jürgen Becker: A Two-level Co-Design Framework for data-driven Xputer-based Accelerators; Proc. of 30th Anual Hawaii Int. Conf. on System Science (HICSS-30), January 7-10, Wailea, Maui, Hawaii, USA, 1997



```
y10 := a0 * (b0 + 2 * c0);
y20 := 5 * d0 + e0 + (f0 + b0);
y30 := g0 * (h0 + 2 * e0);
y40 := (5 * d0 + e0) * f0;
y11 := a1 * (y10 + 2 * c1);
y21 := 5 * y20 + e1 + (f1 + y10):
y31 ;= y30 * (y40 + 2 * e1);
y41 := (5 * y20 + e1) * f1;
```

a)
b)
CoDe-X
(details see figure 8)
DPSS
c)
y10 := a0 * (b0 + 2 * c0);
y30 := g0 * (h0 + 2 * e0);
y40 := (5 * d0 + e0) * f0;
c0
c1
d0
e0
e1
2*
5*
a0
f0
e0
y10
b0
h1
g0
y40
temp
f1
a1
f0
g
y11
y20
e1
f1
y30
y31
bus
y31 := y30 * (y40 + 2 * e1);
y21
y11 := a1 * (y10 + 2 * c1);
temp
y41

Figure 5. Application Example of an automatic and optimized Kress ALU Array Mapping.

- massive run time to compile time migration

The Kress method maps irregular data paths onto a regularly structured platform. Originally Kress has called his approach rDPA (reconfigurable data path array) [13] [15]. The PE (reconfigurable Processing Element) of the Kress Array provides high flexibility by following features.

- 32 Bit PE (reconfigurable Processing Element)
- PE has rALU, small register file, routing resources
- operations, routing only, routing and operations
- routing-only use substantially adds flexibility
- rALU: arithmetic, relational, logic, special, xfer
- pipe-like asynchronous inter-PE communication
- smart interface for data scheduling (data streams entering and leaving the FPAA) [12] [15].

Figure 6 illustrates the routing programmability of the proposed rDPA-II architecture of a Kress Array, which provides non-multiplexed bidirectional interconnect resources between nearest neighbor PEs. The current version Kress Array rDPA-I provides only a single 32 bit channel between nearest neighbor PEs. The example in figure 5 c is based on the rDPA-I Kress Array only [12].

2.1 A Compiler for Structural Software

Like from an ASAP, multiple data streams enter and leave the Kress Array. To organize these data streams, Kress uses *Data Scheduling* [12]. Resource scheduling is no more needed here, since this has been already done by DPSS, the simulated annealing optimizer. Of his Data Scheduling method Rainer Kress has implemented a particular data scheduler, which is supported by the MoM-III Xputer architecture platform and its smart memory interface.

But also a better compiler front end is required. If we need a hardware expert to reconfigure accelerators, the problem description is not really *structural software*. Structural software being really worth such a term would require a source notation like the C language (also see [14]) and an compiler which automatically generates structural code from it. For such a new class of hardware platforms a completely new class of compilers is needed, which generate both, sequential and structural code: partitioning compilers, which separate a source into cooperating structural code (for the accelerator) and sequential code segments (for the host).

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 (rDPA).

Section 4 presents the dual compilation framework and its strategies including the option of using additional data-procedural language features for experienced users who desire hand-honed optimum code. Section 5 discusses computation-intensive application examples from image processing. But before we introduce the target hardware paradigm surrounding the Kress Array, since needed to fully understand the role of the CoDe-X development system.

3. Xputer-based Accelerators

The Xputer machine paradigm (also see figure 7) has been introduced and discussed elsewhere [11] [16] [17] [18]. Xputers use one or more data sequencers — in contrast to (von Neumann) Computers, which use an instruction sequencer. Being procedurally data-driven, the Xputer paradigm is well suited to cooperate with the data-driven Kress Array. Besides, the Xputer paradigm is a useful common model of the wide variety of architectures obtained from CCM implementations and hardware/software co-design efforts: for a completely

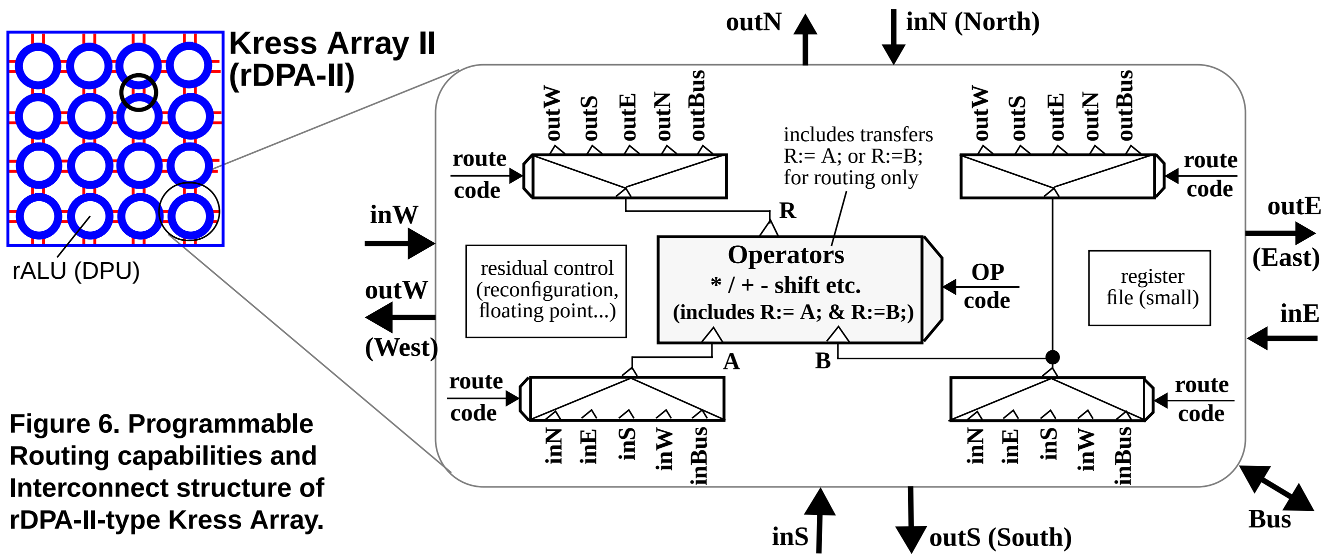


Figure 6. Programmable Routing capabilities and Interconnect structure of rDPA-II-type Kress Array.

different new ALU an Xputer does not need a new sequencer.

The MoM (Map-oriented Machine [19]) is one out of many possible Xputer architectures. Main features of the MoM Xputer architecture are: multiple 2-D data sequencers, 2-dimensional primary memory space, 2-dimensional *scan windows* of arbitrary size and shape, adjustable at run time. Scan windows look at local segments of memory space, and are part of a *smart register file* interfacing primary memory.

Many applications require iterating the same data manipulations on a large amount of data, e.g. statement blocks in nested loops. The Xputer machine paradigm accelerates 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 [11], [16], [20], [21].

3.1 The MoM-III Xputer Architecture

The MoM-III accelerator [22] is an Xputer architecture. It consists of several (up to seven) modules (figure 3), connected to a host computer. Making use of the host simplifies disk access and all other I/O operations. After setup, each 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 concurrently execute other tasks in-between. This allows the use of modules as general purpose acceleration boards for time critical parts in an application. A module consists of three major parts:

- primary memory
- reconfigurable ALU Array including several rALU subnets for multiple scan windows (SW)
- (multiple) generic address generators (GAGs)

The data memory, local on each module, is primarily organized two-dimensional (but can also be n-dimensionally interpreted). To optimize data access by *generic scan patterns* a particular storage scheme is used, called *data map*. The main memory of the host and the local memories are logically a single shared memory (figure 4). The host has access to all local memories, and MoM modules have access to main memory and all other local memories. Additionally, the external bus can be used, but only by a single module at a time.

In accessing arrays and other regular data structures the address sequence(s) can be specified by a parameter set, from which the reconfigurable GAGs [23] carry out generic 2-dimensional movement patterns of the *scan windows* which are called *generic scan patterns* [11]. Thus GAGs contribute to speed-up by avoiding addressing overhead, since GAGs compute generic scan patterns without needing any memory cycles other than for initial parameter fetch.

The MoM-III data sequencer is implemented with reconfigurable logic. The control part is admitted with features for reconfiguration and realized with a single Xilinx XC4013 FPGA. Because of the large word length, such a fine grained structure is not suitable for the datapath of the GAG. Therefore the reconfigurable Kress Array is used [24].

In the Kress Array arithmetic, relational, logic, and other operators up to 32 bits wide can be implemented in a single logic datapath unit (DPU), faster and more area-efficient than by FPGAs [15]. Our current Kress Array prototype has 72 identical word-oriented DPUs, running structural code compiled from C programs. See mapping example in figure 5 c.

All global I/O operations are done by the host by taking advantage of the functionality of the host's operating system. The host has direct access to all memories on the modules via a small bus interface (IF, see figure 3). The MoM can access data in the host's memory. The MoM activation as well as its synchronization and data transfers with the host are controlled by the MoM Xputer Run Time System (XRTS), which is

von Neumann Computer	**Xputer**	Data Flow Machine
control-driven	data-driven	
deterministic (procedural)		indeterministic (arbitration-driven)

Figure 7. Comparing the 3 basic Machine Paradigms

explained in section 4.1.

The Xputer paradigm provides several advantages: complex compound operators can be configured to multiple and non-linear pipelining with high instruction level parallelism, which is impossible with fixed instruction set processors. Intermediate results can be passed along in the pipeline, instead of writing them back into register file or memory after each instruction execution, what drastically accelerates by saving a lot of memory cycles. Furthermore most addressing and control overhead is migrated from run time to compile time, due to GAG use [24] giving hardware support for a rich repertory of data access sequences [11], [16]. A major advantage of the Kress Array (figure 2) is its flexibility, so that also the data paths of GAGs can be mapped onto it [24], so that only one major chip design is needed for an Xputer.

4. Dual Compilation Framework CoDe-X

Empirical studies of computation-intensive applications [25] conclude that the major amount in computation time is due to rather simple loop constructs. Since often these loops reference indexed array data structures, von Neumann computers are burdened with massive addressing overhead. First efforts to reduce addressing overhead and to introduce parallelism have lead to the development of pipelined and vector supercomputers [26]. In this context parallelizing compilers have been developed, based on data dependence analysis [20], [27] and producing executable code for different parallel target machines. Unfortunately, the hardware structures do not reflect the data structures of most algorithms very well, which substantially restricts the exploitation of inherent parallelism.

For the MoM a partitioning and parallelizing compilation framework CoDe-X is being implemented, based on two-level hardware/software co-design strategies and accepting X-C source programs (figure 8). X-C (Xputer-C) is a C dialect. CoDe-X consists of a 1st 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 MoM (Xputer tasks). Parts for MoM execution are expressed in a C subset, which lacks only dynamic structures [28] [29]. 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.

By using C extensions within X-C experienced users may

The Xputer paradigm is for structural programming, what von Neumann is for procedural programming.

The Xputer paradigm is for soft machines, what von Neumann is for hardwired ones.

hand-hone their source code by including directly data-procedural MoPL code (Map-oriented Programming Language [30]) into the C description of an application. Also less experienced users may use special MoPL library functions similar to C function calls to take full advantage of the high acceleration factors possible by the Xputer paradigm (see figure 8, section 4.2 and section 5). In next sections the profiling-driven host/Xputer partitioning is explained first. Then the data-procedural options are discussed, and finally the integration of the X-C subset compiler by Karin Schmidt [28], [29] into the CoDe-X framework (2nd partitioning level) is described.

4.1 Profiling-driven Host/Accelerator Partitioning

The first level of the partitioning process is responsible for the decision which task should be evaluated on the MoM Xputer and which one on the host. Generally three kinds of tasks can be determined:

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

The host tasks have to be evaluated on the host, since it cannot be performed on the MoM accelerator. The MoM tasks are the candidates for the performance analysis on the host and on the MoM. The MoM Xputer library functions are used to get the highest performance out of the MoM. An experienced user has developed a function library with all 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.

MoM Xputer tasks are candidates for the partitioning process in the first level. The user may advise the partitioner about parts to be MoM 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 MoM execution. Such a MoM job is divided into the following MoM tasks, which can be computed on a single MoM module:

- basic block (a linear code sequence),
- basic block including conditions,
- fully nested loops (up to seven loops), or
- 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 (see figure 10).

The restriction to seven nested loops is due to limited

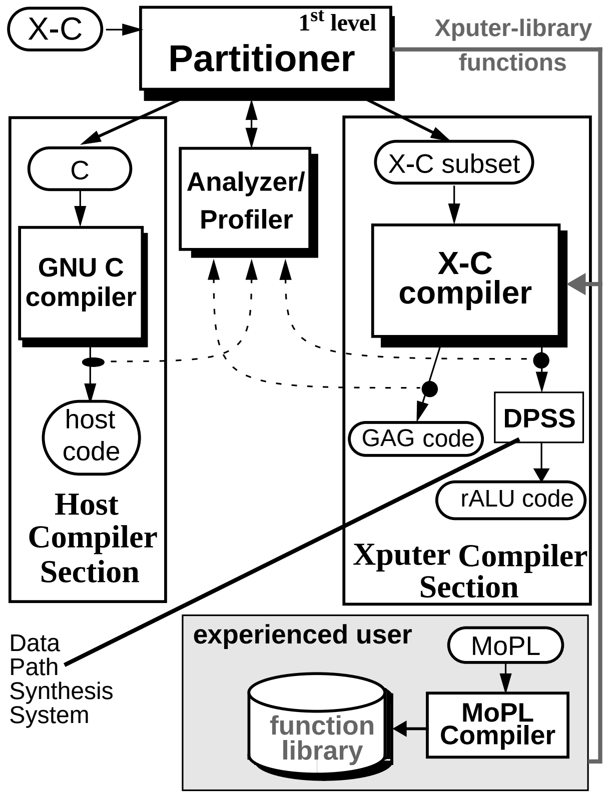


Figure 8. Overview on the CoDe-X Framework

resources in the current MoM prototype. The data sequencer of this prototype contains seven GAGs, which may run in parallel. A MoM job is represented by a task graph, on which a data dependency analysis based on the GCD-test [40] is carried out. Thus tasks which can be evaluated in parallel can be found. In this phase several code optimization techniques are possible in order to exploit the target hardware in an optimized way.

For example, single nested loops of large tasks can be transformed into double nested loops by organizing the computation in the original loop into chunks of approximately equal size. This transformation is called strip mining [31] (see figure 9), which makes it possible to compute these chunks in parallel on different MoM Xputer modules. In our approach the block size of the chunks depends on parameters describing the hardware resources of the current MoM prototype in order to achieve an optimized performance/area trade-off. This technique can be used e.g. in image processing applications by dividing an image in stripes of equal sizes in order to manipulate these stripes concurrently [32]. The optimization techniques [31] of loop distribution (splitting of one loop into two loops computing the same), of loop fusion (transforming two adjacent loops into a single loops) and of loop interchanging (switching inner and outer loop) are also performed by the 1st level partitioner in order to exploit potential parallelism and to optimize the utilization of the MoM hardware resources.

But there may be two problems: if the accelerator needs

```
Do I = 2 to N
   A [I] = B [I] + 1;
   D [I] = B [I] - 1;
Enddo;
```

```
Do J = 1 to N by 32
   Do I = J to min (J+31, N);
     A [I] = B [I] + 1;
     D [I] = B [I] - 1;
   Enddo;
Enddo;
```

Figure 9. Strip mining example (block size: 32)

reconfiguration at run-time and in a few cases, if the datamap has to be reorganized also at run-time. Independent from other active MoM modules run-time reconfigurations can also be performed partially and datamap reorganizations are done by the host [32].

Synchronization and Run Time Reconfiguration. The MoM Xputer Run-time System (XRTS) [33] provides the software-interface from the host to MoM-based accelerators. The main purpose of this system is controlling and synchronizing applications executed on the accelerator, but additional features like program debugging are also possible. The MoM Xputer Run-time System can be started interactively or inside a host process and has following features:

- XRTS loads MoM Xputer object files (sequential and structural code for GAGs and rALU Array)
- XRTS loads application data binary files with optimized distribution to the memories of different modules (figure 3) to minimize inter-module communication

During applications sometimes reconfigurations are necessary: data sequencers need new parameter sets, and the rALUs require reconfigurations. The parameter set of the data sequencer can be neglected since there are only a few parameters to configure. The configuration of the rDPA Kress Array requires 147456 bits in the worst case and 23040 bits in the average case. Such a number cannot be neglected. In any case it is useful to have a local reconfiguration memory on each Xputer board. There are three possibilities to reduce the overhead for reconfiguration at run-time:

- Configure an idling board while other boards are running
- Partial configuration: code to be loaded partially only
- Context switch between resident configuration sets: commercially available only with Xilinx XC6200 series.

This synchronization and reconfiguration overhead is considered during the iterative first level partitioning process, where each iteration estimates the overall execution time of actual applications, to be optimized finally (see equation (1)).

Datamap Reorganization. Due to the large repertory of

Reiner W. Hartenstein, Jürgen Becker: A Two-level Co-Design Framework for data-driven Xputer-based Accelerators; Proc. of 30th Anual Hawaii Int. Conf. on System Science (HICSS-30), January 7-10, Wailea, Maui, Hawaii, USA, 1997

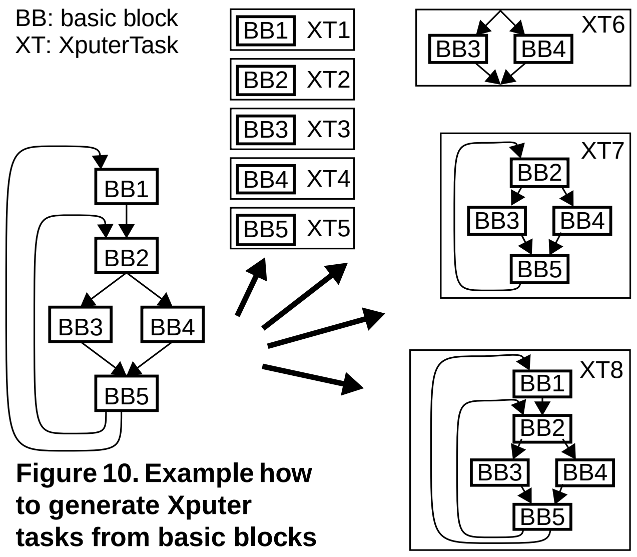


Figure 10. Example how to generate Xputer tasks from basic blocks

scan patterns of the generic address generators (GAGs), a reorganization during run-time of the storage schemes (data maps) in the memories of Xputer modules is not required in many cases. The first level partitioner is analyzing the generated storage schemes of variable arrays in all Xputer tasks and includes necessary code for reorganizing data maps during run-time into the code of host tasks. A necessary reorganization can be performed by the host since the host has direct access to all memories of Xputer modules via the bus interface. When the host evaluates a task, it can read from any local memory and may also write back to any local memory. The corresponding run-time overhead is also considered in equation (1) during the iterative first level partitioning process. For further details of datamap reorganization during first level partitioning see [34].

Profiling. For each Xputer task, the worst case execution time on the accelerator as well as on the host is computed. The performance data for the accelerator is received from the X-C compiler by analyzing the execution time of the compound operators and their iterations [12], [32].

The performance of the host is approximated by examining the code. For each processor type and workstation, another model is required. The behavior of the underlying hardware and operating system has to be deterministic and known. This implies the timing behavior of all hardware components, the effects of caching, pipelining, etc. The operating system must provide static memory management, system calls should have a calculable timing behavior, and no asynchronous interrupts should be allowed [35]. The techniques used are similar to these from Malik et. al. [36].

The profiler is also computing the overall execution time t_{acc} by using the Xputer as accelerator. The time t_{acc} includes delay times for synchronization, possible reconfigurations and memory re-mappings during run time (see equation (1)).

$$t_{acc} = \sum_{i \in HT} t_{exe_i} + \sum_{j \in XT} t_{exe_j} + \sum_{j \in XT} t_r + \qquad \text{(eq. 1)}$$

whereas:

$\sum_{i \in HT} t_{exe_i}$: sum of execution times of tasks executed on the host (HT)

$\sum_{j \in XT} t_{exe_j}$: sum of execution times of tasks executed on the Xputer (XT)

$\sum_{j \in XT} t_r$: sum of delay times for reconfiguring the Xputer during run time

$\sum_{j \in XT} t_{mem_j}$: sum of delay times for re-mapping the 2-dimensional organized Xputer data map during run time

$\sum_{\forall XputerCalls} t_{syn}$: sum of delay times for synchronizing host/Xputer (operating system calls)

$\sum_{\forall XT} t_{ov}$: sum of overlapping execution times between tasks executed simultaneously on host/Xputer

The overall execution time is determined by delay times of the tasks (host and Xputer), plus the above mentioned penalty times (see t_{acc}, equation (1)). As mentioned above memory re-mappings are necessary, when two tasks use the same data onto the Xputer, but they are needing a different distribution of the data within the two-dimensionally organized memory. Then a re-ordering of the data items must be performed before the execution of the second task starts, which must be also considered in equation (1). The value of t_{acc} is used as cost function in the simulated annealing process [37], [32] and has to be optimized by computing different partitionings of Xputer tasks in varying their task allocation between host and Xputer.

Simulated Annealing. The iterative partitioning phase is based on a simulated annealing algorithm [37]. Our algorithm computes different partitionings of Xputer tasks by varying their task allocation between host and Xputers. As mentioned above the initial partitioning is a resource-constraint list-based scheduling of the task graph, where the data dependencies are considered. The Xputer tasks of the initial partitioning are swapped. If the new partitioning results in a lower overall cost, it is accepted. If not, there is still a finite probability to accept. This probability depends on the temperature of the annealing process. This avoids that the algorithm gets stuck in a local minimum. Since a good initial partitioning is used at the beginning, we start with a low initial temperature. The temperature is gradually lowered fast at the beginning and slower at the end. For further details about the simulated annealing process during first level partitioning see [32].

4.2 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

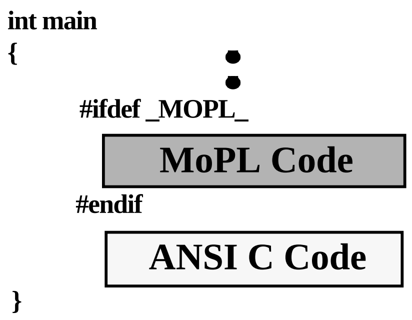


Figure 11. Example of including a MoPL code directly into an X-C program

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 [30] fully supporting all features of the Xputer hardware.

The input specification is then compiled into Xputer code, namely the datamap, the data sequencer code and the rALU code, dependent on the concrete function 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 12). In this example a 8 by 10 matrix is multiplied 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 12). 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 [30]) 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 12), 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 5.

4.3 Resource-driven second level Partitioning

The X-C compiler realizes the 2nd 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 [38] resulting in a partial execution order. This algorithm is partitioning the

```
    #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 12. Example of including a data procedural function call into an X-C program

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 avoiding reconfigurations or idle hardware resources [28]. Xputers provide best parallelism at statement or expression level. So we try to vectorize the statement blocks in nested for-loops according to the vectorization algorithm of J. R. Allen and K. Kennedy [39], after a data dependency analysis has been performed [40].

The access sequences of up to 4 fully nested loops contain variables with linear index functions can be handled by transforming them into parameters for configuring the GAGs (see figure 3). A configuration file for the X-C compiler gives the current restrictions of the hardware, e.g. number of GAGs available, size of rALU subnets etc., which allows a flexible handling of the compiler for future up-grades of the Xputer hardware. Further details and examples about the X-C compiler can be found in [12], [28], [29].

5. Examples: Image Processing Applications

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:

$$g(x,y) = \frac{1}{M}\sum_{(n,m)\in S} f(n,m) \qquad \text{(eq. 2)}$$

for x,y = 0,1, ..., 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 every 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 1st 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 4.3). The final acceleration factor in comparison with a SUN SPARC 10/51 was 73. For details about this example see [32].

Another application from the area of image processing are sharpening algorithms, which are special cases of the discrete convolution for detecting edges. In figure 13 a data procedural function named "SharpImage" is called within an X-C program performing a sharpining over a 640 by 400 pixel image by using a 3 by 3 coefficient matrix. This Xputer library function will be executed in every 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 13 the MoPL-code is outlined only briefly, for details about prorgamming with MoPL please see [30]. If no library function for sharpening would have been available, the MoPL-code in figure 13 could also be included directly into the X-C program by experienced users.

```
    #include <stdio.h>
    #include <MoPL.h>
    int Im[640][400][32];
    int Coeff[3][3][32];
                          ⋮
/* Xputer-library function call for sharpening a 640 x 400
   pixel image with a 3 by 3 coefficient matrix */

    SharpImage (Im,Coeff,640,400,3,3);
                          ⋮
}                  |   Transformation of library
  {                ↓   function into MoPL code
                          ⋮
    #ifdef _MOPL_
     mopl sharpening (ref int Im[640][400][32],
                      val int Coeff[3][3][32]
     array Im     [1:640, 1:400] of int;
           Coeff   [1:3, 1:3] of int;
     scanPattern  VideoScan is ...
     window SharpenW is ...
     rALUsubnet Sharpen of SharpenW is ...
     begin /* main MoPL-program part */
       with SharpenW do begin
        apply Sharpen;
        move SW1 to Im [1,1];
        move SW2 to Coeff [1,1];
        VideoScan [SW1];
                          end;
     end;
    #endif
                          ⋮
  }
```

Figure 13. Transformation of a data procedural function into corresponding MoPL-code

6. Conclusions

A novel compilation environment and a powerful general purpose reconfigurable hardware platform for software-only accelerator implementation of highly computation-intensive applications has been introduced. The reconfigurable Kress ALU array and the Xputer machine paradigm have been summarized. The Kress method permits to map software-only accelerator implementations onto a fast full-custom general purpose technology platform of high layout density, where only a single major design is needed.

The partitioning and parallelizing compilation system CoDe-X for this technology platform has been presented, which accepts C language source programs, and, which generates sequential code for the host, as well as structural code, data schedules and storage schemes for MoM accelerator Xputers equipped with Kress ALU arrays.

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 [32].

7. Call for Partners

Since in Europe we do not have a MOSIS service and VLSI prototyping infrastructures for universities are declining and meanwhile rotten, we do not feel to be strong as a VLSI design crew. So we fear not to be able to implement the newest version Kress Array at a reasonable turn-around time. That's not the only reason, why we are looking for partners. Contact us!

We have a lot of know-how on all aspects of data-driven reconfigurable general purpose accelerators, and its technology platforms, applications, application development support tools, (programming) languages etc. Since, disappointed by the various parallel computing scenes, funding agencies switch over to reconfigurable systems, the time has come. For this promising R&D area the break-through seems to be near. Academic and commercial merits of fundamental significance are waiting for those, joining this new motivating R&D area.

8. References

[1] R. Hartenstein, (key note): Custom Computing Machines - An Overview; Workshop on Design Methodologies for Microelectronics, Smolenice Castle, Slovakia, Sept. 1995

[2] 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 (also in [3])

[3] R. Hartenstein, M. Glesner: Field-Programmable Logic - Smart Applications, New Paradigms and Compilers; Springer Verlag 1996, (Lecture Notes on Computer Science, vol. 1142)

[4] D. Monjau: Proc. GI/ITG Workshop Custom Computing, Dagstuhl, Germany, June 1996

[5] D. Buell, K. Pocek: Proc. IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, Napa, CA, April 1993

[6] see [5], but 1994, 1995, and 1996

[7] R. Ernst, J. Henkel, T. Benner: Hardware-Software Cosynthesis for Microcontrollers; IEEE Design & Test, Dec. 1993

[8] R. K. Gupta, C. N. Coelho, G. De Micheli: Program Implementation Schemes for Hardware-Software Systems: IEEE Computer, pp. 48-55, Jan. 1994

[9] 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 (see [6])

[10] R. Hartenstein, J. Becker, M. Herz, R. Kress, U. Nageldinger: A Synthesis System for Bus-based Wavefront Array Architectures; Int'l Conf. on Application-specific Systems, Architectures, and Processors (ASAP'96), Chicago, Aug, 1996, IEEE CS Press 1996

[11] R. Hartenstein, A. Hirschbiel, M. Weber: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware; InfoJapan'90- Int'l. Conf. memorizing the 30th Anniv. of Computer Society Japan, Tokyo, Japan, 1990;

[12] R. Hartenstein, R. Kress: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; ASP-DAC'95, Makuhari, Chiba, Japan, Aug. 29 - Sept. 1, 1995

[13] R. Kress: Computing with reconfigurable ALU arrays; IT Press Verlag, Bruchsal / Chicago (planned for 1997)

[14] K. Tammemäe, M. O'Nils, A. Hemani: Flexible Co-Design Target Architecture for Early Prototyping of CMIST Systems; 6th Int'l. Workshop on Field-Programmable Logic and Applications (FPL'96), Darmstadt 1996 (also in [3])

[15] R. Kress: A fast reconfigurable ALU for Xputers; Ph. D. dissertation, Kaiserslautern University, 1996

[16] see [11]: invited reprint in: Future Generation Computer Systems no. 7, pp. 181-198 (North-Holland, 1991/92)

[17] R. Hartenstein: FAQ and FQA about Xputers; see http:// xputers.informatik.uni-kl.de/

[18] B. Fawcett, J. Watson: Reconfigurable processing with Field Programmable Gate Arrays; Int'l Conf. on Application-specific Systems, Architectures, and Processors (ASAP'96), Chicago, Aug, 1996, IEEE CS Press 1996

[19] R.W. Hartenstein, A.G. Hirschbiel, M. Weber: A Flexible Architecture for Image Processing; Microprocessing and Microprogramming,vol 21, pp 65-72, North-Holland, 1987

[20] U. Banerjee: Dependence Analysis for Supercomputing; Kluwer, 1988.

[21] R. 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

[22] R. W. Hartenstein, M. Riedmüller, K. Schmidt, M. Weber: A Novel Asic Design Approach Based on a New Machine Paradigm; Special Issue of IEEE Journal of Solid State Circuits on ESSCIRC´90, July 1991

[23] R. Hartenstein, H. Reinig: Novel Sequencer Hardware for High-Speed Signal Processing; Int'l Workshop on Design Methodologies for Microelectronics, Smolenice Castle, Sept. 1995

[24] R. Hartenstein, J. Becker, M. Herz, R. Kress, U. Nageldinger: A Partitioning Programming Environment for a Novel Parallel Architecture; 10th International Parallel Processing Symposium (IPPS'96), Honolulu, Hawaii, April 1996

[25] Z. Shen, Z. Li, P.-C. Yew: An Empirical Study of Fortran Programs for Parallelizing Compiler; IEEE Trans. on Parallel and Distributed Systems **1**, 3, pp. 356-364, July 1990.

[26] K. Hwang: Advanced Computer Architecture: Parallelism, Scalability, Programmability; McGraw-Hill, 1993.

[27] H.P. Zima, B. Chapman: Supercompilers for Parallel and Vector Computers; Frontier Series, Addison-Wesley, 1990

[28] K. Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph.D. Thesis, Kaiserslautern 1994

[29] K. Schmidt: Xputers: a high performance computing paradigm; IT Press (planned for 1997)

[30] A. Ast, J. Becker, R. 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, Springer, 1994

[31] 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

[32] 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

[33] U. Nageldinger: Design and Implementation of a menu-driven Run Time System for the MoM-3; Master Thesis, University of Kaiserslautern, 1995

[34] J. Stumpe: Comparison and Reorganizing the storage schemes of variable arrays in the Xputer Multi Tasking Mode; Project Thesis, University of Kaiserslautern, 1996

[35] 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

[36] 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

[37] N. A. Sherwani: Algorithms for Physical Design Automation; Kluwer Academic Publishers, Boston 1993

[38] R. E. Tarjan: Testing Flow Graph Reducibility; Journal of Computer and Systems Sciences 9 (1974), pp. 355-365

[39] K. Kennedy: Automatic Translation of Fortran Programs to Vector Form; Rice Technical Report 476-029-4, Rice University, Houston, Oct. 1980

[40] M. Wolfe, C.-W. Tseng: The Power Test for Data Dependence; IEEE Transactions on Parallel and Distributed Systems, Vol. 3, No. 5, Sept. 1992