Reiner W. Hartenstein, Jürgen Becker, Michael Herz, et al.: A Parallelizing Programming Environment for Embedded Xputer-based Accelerators; High Performance Computing Symposium '96, Ottawa, Canada, June 1996

A Parallelizing Programming Environment for Embedded Xputer-based Accelerators

R. W. Hartenstein, J. Becker, M. Herz, R. Kress, U. Nageldinger
University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de

Abstract

This paper presents the parallelizing programming environment CoDe-X introducing hardware/software co-design strategies on two levels for Xputer-based accelerators. Xputers are based on field-programmable logic providing high level of parallelism for the computation of data and on hardwired data sequencers providing accessing-sequences for fast address computation. CoDe-X accepts a C dialect, also including a data-procedural extension, and carries out both, the profiling-driven host/accelerator partitioning, and (2nd level) the resource-parameter-driven sequential/structural partitioning of the accelerator source code to optimize the utilization of its reconfigurable datapath resources.

1. Introduction

From empirical studies [32] it can be concluded that the major amount in computation time is due to rather simple loop constructs. Studying DSP and image preprocessing algorithms on a von Neumann platform we found examples where up to 94% of the execution time elapsed for address computation overhead. Parallelizing compilers have been developed, where compilation is based on loop transformations [36]. But the hardware structures are not reflecting the structure of the algorithms very well, which substantially restricts the exploitation of inherent parallelism.

Due to the availability of field-programmable logic (FPL), a new programming paradigm has been arisen, the structural programming paradigm. Algorithms or parts of algorithms are ‘compiled’ onto hardware rather than be evaluated from software. Accelerators based on such reconfigurable hardware are called custom computing machines (CCMs: [13] [16], [25]). 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 [5], [8], [22]. Further co-design approaches are advocated explicitly or implicitly in developing custom computing machines such within the R&D scenes of ASIPs [4] [24] [29] and FCCMs [26] [27]. With the FCCM approach a von Neumann host implementation is accelerated by external field-programmable circuits. Here application development usually requires hardware experts. The von Neumann paradigm 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] [12] or the rDPA approach (reconfigurable data path array) [19] [20].

For such a new class of hardware platforms a new class of compilers is needed, including partitioning and parallelizing compilers, to link Xputers as universal embedded accelerators to workstations or other hosts. This paper presents a compilation framework based on this paradigm. The hardware/software co-design framework CoDe-X targets the partitioning and compilation of a C dialect, called X-C, onto a host using the Xputer as universal configurable accelerator. X-C (Xputer-C) is a C dialect, also including a data procedural extension, where experienced users may experiment with the Xputer data sequencing primitives to learn, how to obtain optimum acceleration advantage from this paradigm. At first level, per-

formance-critical parts of an algorithm are localized and migrated to the Xputer without manual interaction. Profiling-driven strategies for synchronization overhead reduction, run-time reconfiguration, and memory organization are discussed. The second partitioning level performs a resource-parameter-driven procedural/structural partitioning of (Xputer) accelerator source code by generating both: structural code for configuring both: the address generators as well as the reconfigurable datapath resources, and, sequential code for (the) data sequencer(s). The goal is to optimize the utilization of the Xputer hardware resources providing parallelism at instruction level. In contrary to hardware/software co-design systems like Cosyma [5] or Vulcan [8], here the problem of accelerators based on a new machine paradigm is discussed. The fundamentally new feature of the CoDe-X framework is the two-level co-design approach.

First this paper describes the underlying target hardware. Section 3 presents the co-design framework CoDe-X and its strategies. Section 4 discusses a computation-intensive application example from the area of image processing.

2. Xputer-based Accelerators

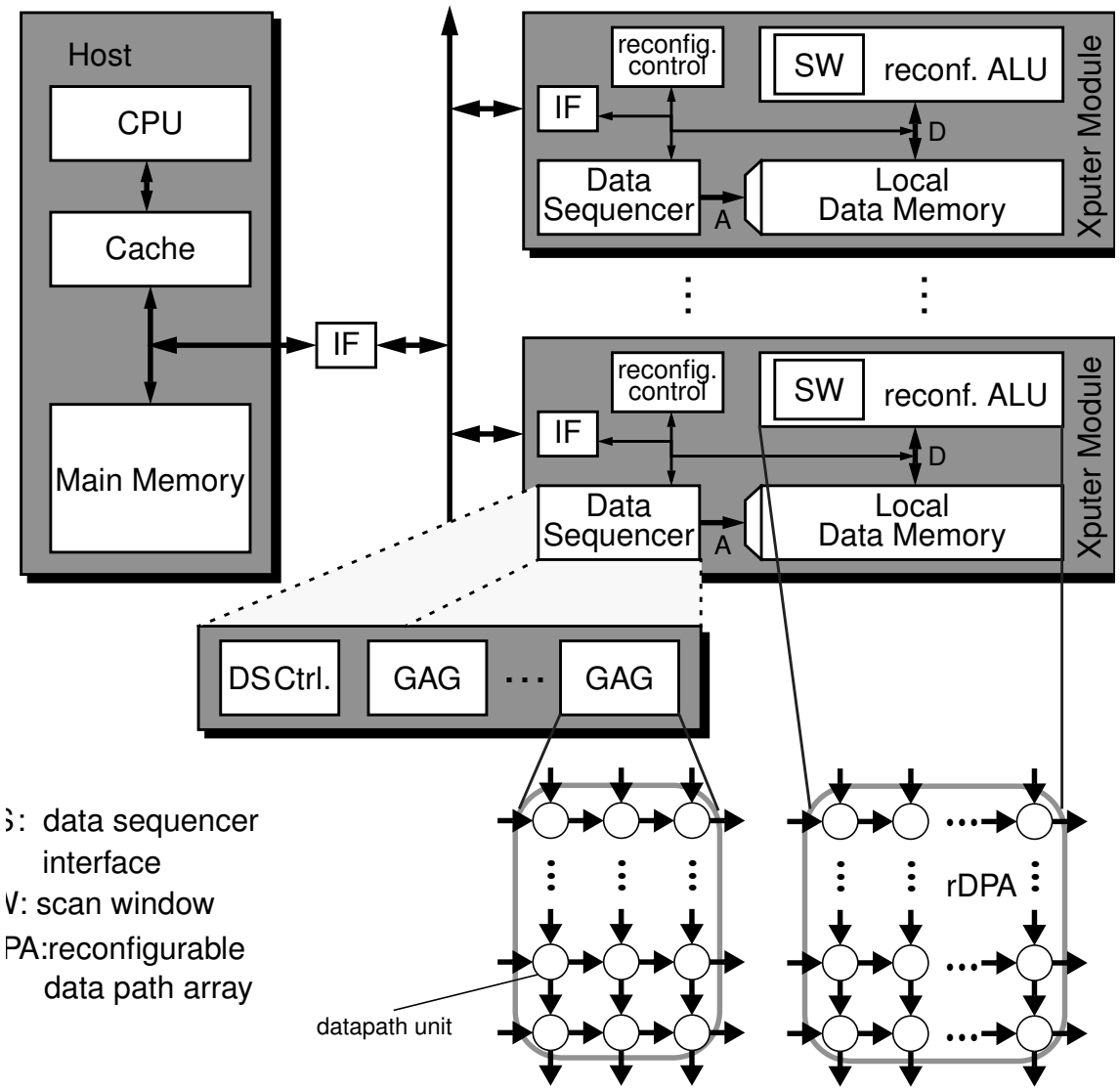
Many applications require the same data manipulations to be performed on a large amount of data, e.g. statement blocks in nested loops. The Xputer machine paradigm aims at the acceleration of such applications. Xputers are especially designed to reduce the von Neumann bottleneck of repetitive decoding and interpreting address and data computations. High performance improvements have been achieved for the class of regular, scientific computations [3] [9] [11].

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 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 and logic unit (rALU) including several rALU subnets with multiple scan windows
- the data sequencer (DS) comprising several generic address generators (GAGs)

Figure 1. Host with Xputer-based accelerator comprising several modules

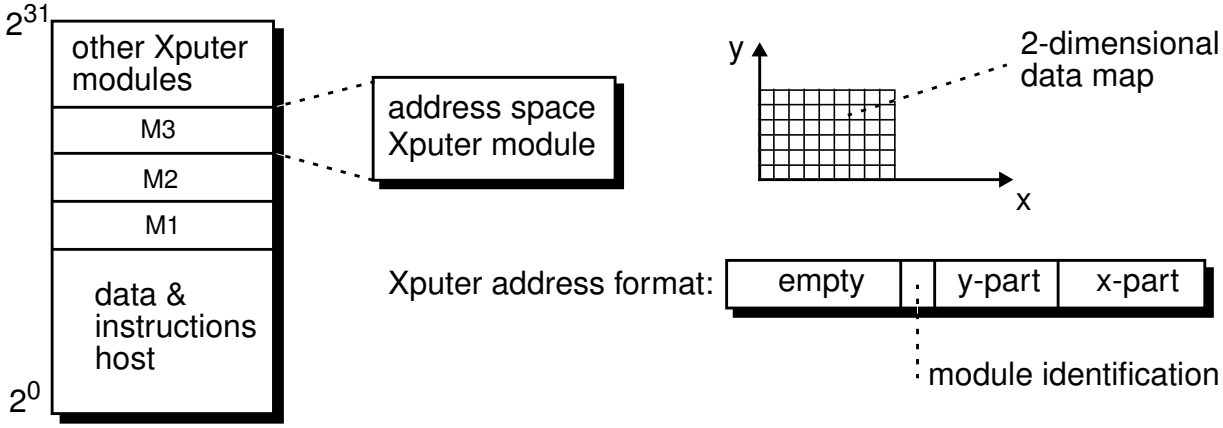


The data memory which is local on each Xputer module, is primarily organized two-dimensional, but can also be interpreted as higher dimensional. It contains the data which has to be accessed or modified during an application. The data is arranged in a special order to optimize the data access sequences. This arrangement is called data map. The scan windows (SW) serve as interface of the rALU subnets to the data memory. A scan window holds a copy of the data out of a local neighbourhood of the data map. Scan windows are adjustable in size during run-time of an application. The main memory of the host and the local memories are logically a single shared memory (figure 2). The host has access to all local memories, and vice versa the Xputer modules have access to the main memory as well as to other local memories. Of course local memory access is much faster than remote access to others via the bus. Additionally, the external bus can be used only by a single Xputer-module at a timeA 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 [14]. The address sequences are called scan patterns. 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. Data sequencing in general means, that the GAGs address data at correct locations in the data memory by generic scan patterns and load it into the scan windows of the rALU subnets.

All data manipulations are done in the reconfigurable ALU (rALU). The rALU consists of several rALU subnets. Each subnet is distinguished by its rALU subnet number. Each Xputer-module can simultaneously use three different rALU subnets on its board. The rALU subnets perform the computations on the data which are provided by the scan windows by applying a compiler configured complex operator on that data. That complex operator is called compound operator. Each rALU subnet has four output flags to control the generic address generators (GAGs), e.g. for data-dependent scan patterns. The operations in the rALU are done in parallel to the address computations of the GAGs. In order to have a general purpose interface between the rALU and the GAGs, these components are transport-triggered, which means the availability of data triggers the operation. This allows to be very flexible in implementing different rALU subnets, as the subnets do not need to have the same computation times. Furthermore, it allows to implement a pipelined rALU concept as well as a combinatorial net. After finishing, the rALU signals the end of the computation to the GAGs

Figure 2. Shared memory structure and Xputer address format.



As long as a rALU operates on the local memory of the same Xputer module, the data accesses can be done in parallel to the other Xputer modules. A non local data access has to be done sequentially via the external bus. Using this bus, the data sequencer can also access data in the host memory. The control unit in the data sequencer is responsible for the configuration of the GAGs. The reconfiguration control unit holds the parameter sets and the configuration data for the complete application in its memory.

For the evaluation of standard C programs, word-oriented operators such as addition, subtraction, or multiplication are required. This realization of a reconfigurable architecture for word-oriented operators needs a more coarse grained approach for the logic block architecture of the rALU. The granularity is more at operator level instead of gate level like in commercially available FPGAs. Complete arithmetic and logic operators can be implemented in a single logic block. Thus, in contrary to FPGAs, such a block is called datapath unit (DPU) to show its prior functionality. The datapath unit can be optimized to implement operators faster and more area efficient than FPGAs.

A word-oriented structure for the datapath unit is no drawback since random logic or control logic need not to be supported by this architecture. Furthermore, such a structure requires less configuration bits which saves additional area. In addition the timing is more predictable since there are less transits of signals via programming switches. This greatly simplifies the synthesis system since it saves a necessary

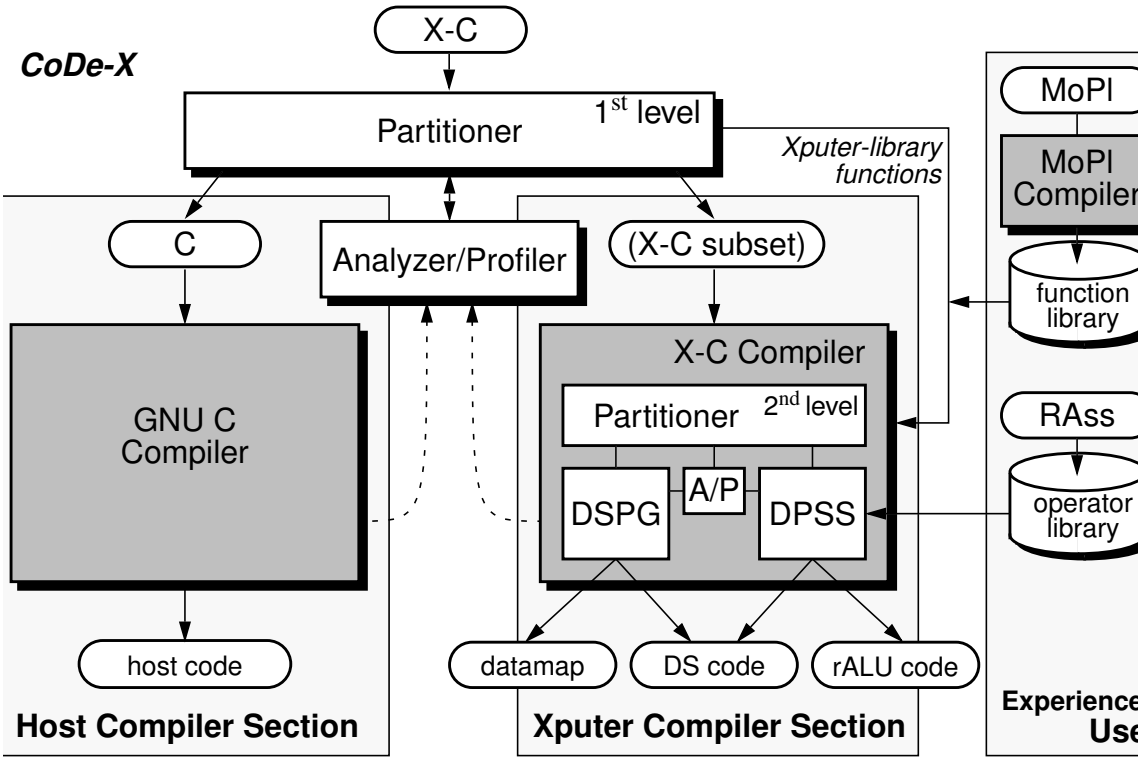
back-annotation step to determine the exact delay times for simulation. In the current prototype, the reconfigurable datapath architecture (rDPA) serves as rALU. It consists of 72 identical word-oriented DPUs. For more details about the rDPA see [3] [10] [19] and [20].

The Xputer paradigm provides several advantages: The combinations of several operations to one compound operator allow to introduce pipelining and fine grain parallelism to a larger extend, as this can be done in fixed instruction set processors. Intermediate results can be passed along in the pipeline, instead of writing them back into the register file after every instruction. Xputers support application specific instruction sets with their reconfigurable ALUs. Furthermore most addressing and control overhead is eliminated by the data sequencer which gives hardware support for a rich repertory of data access sequences [14]. One major advantage of the rDPA is its flexibility, e. g. the data paths of the GAGs can be also synthesized into a rDPA device [15] (see figure 1).

3. The Parallelizing Compilation 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 3). 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 Xputer (Xputer tasks). Parts for Xputer execution are expressed in a X-C subset, which lacks only dynamic structures, but includes language features to express scan patterns [1], [30]. At second level this input is partitioned by the X-C compiler in a sequential part for the DS, and a structural part for the rDPA. Experienced users may use special MoPL library functions [1] to take full advantage of the high acceleration factors possible by the Xputer paradigm (see figure 3). First the profiling-driven host/Xputer partitioning is explained. Then the programming of the Xputer-accelerator is shown and finally the options for experienced users are discussed

Figure 3. Overview on the CoDe-X programming environment.



3.1 Profiling-driven Host/Accelerator Partitioning

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

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

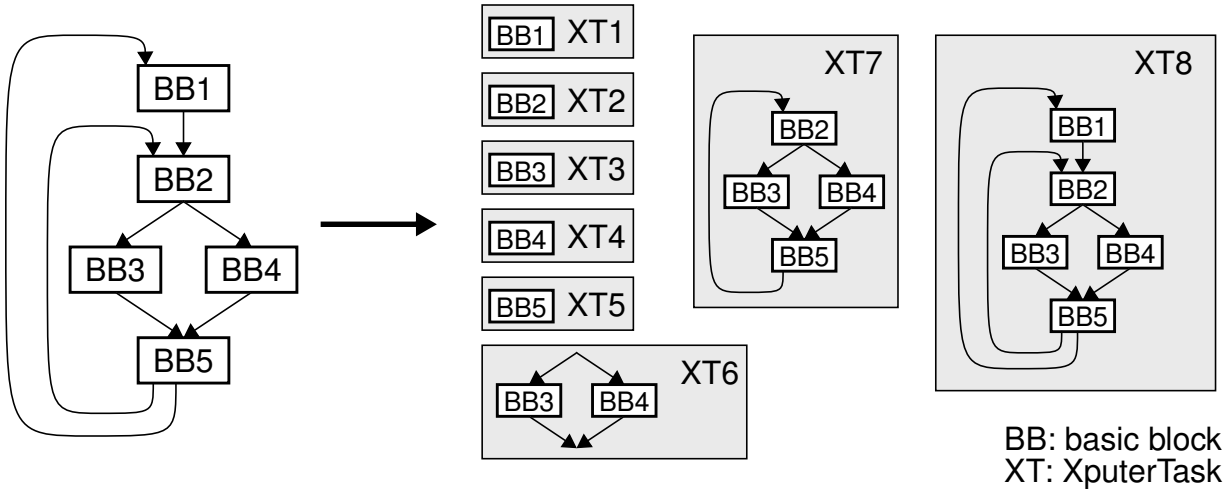
The host tasks have to be evaluated on the host, since it cannot be performed on the Xputer accelerator. The Xputer tasks are the candidates for the performance analysis on the host and on the Xputer. The Xputer-library functions are used to get the highest performance out of the Xputer. An experienced 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, which can be computed on a single Xputer 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 4)

Figure 4. Example how to generate Xputer tasks from basic blocks.



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. An Xputer job is represented by a task graph, on which a data dependency analysis based on the GCD-test [35] is carried out. Thus tasks which may run in parallel are found. In this phase parameter-driven transformations are carried out in order to exploit best hardware utilization. For example, single nested loops of large tasks are transformed into double nested loops (called strip mining [21]), which permits parallel execution on different Xputer modules. The parameters describe the given hardware resources in order to achieve optimized performance/area trade-offs.

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. The optimization techniques [21] 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 Xputer hardware resources.

For an initial partitioning, the host tasks are performed on the host, the Xputer tasks and the Xputer-library functions are performed on the Xputer modules. If several alternative paths are in the task graph, the largest task in the critical path, which fits onto an Xputer module is chosen next to be scheduled. This method performs a resource-constraint list-based scheduling of the task graph [6] and results in a fine initial partition since it can be assumed that an Xputer tasks can be evaluated faster on the accelerator than on the host. But there are two problems: the accelerator needs reconfiguration at run-time and in a few cases, the datamap has to be reorganized also at run-time.

Run-Time Reconfiguration. In the case of an Xputer-based accelerator, the data sequencer requires a new parameter set, and the rALU requires a reconfiguration. The parameter set of the data sequencer can be neglected since there are only a few parameters to configure. The configuration of the rDPA 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 one idle Xputer board while one or several other boards are running. Unfortunately, this is not always possible due to data dependencies.
- Partial configuration: Since the rDPA allows a partial configuration, only the difference between the current and the set to be configured has to be loaded.
- Context switch: A context switch allows the switching between two or more already loaded configuration sets. In the case of the rDPA array 2304 bits have to be transmitted to switch all 72 datapath units. Only the new Xilinx XC6200 series provides such a context switch too.

Datamap Reorganization. Due to the large repertory of scan patterns of the generic address generators (GAGs), a reorganization of the datamap in the local memories is not required in many cases. A necessary reorganization can be performed by the host since the host has direct access to the local memories via the external bus and the interface. When the host evaluates a task, it can read from any local memory and may also write back to any local memory.

Pro[illegible] 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 datapath synthesis system (DPSS) in the X-C

compiler. The X-C compiler is able to evaluate the number of iterations for the compound operator. Knowing this execution time, the complete evaluation time can be approximated by multiplying it with the iteration count. The time-consuming placement of operators in the DPSS does not have to be performed for this analysis.

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 [28]. Branch prediction techniques [2] are considered by computing the execution time of a code segment. The techniques used are similar to these from Malik et. al. [23].

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 of the Xputer during run time (see equation (1)). The iterative partitioning process tries to optimize t_{acc} and is based on simulated annealing, which is described later in this chapter.

(eq. 1)

$$t_{acc} = \sum_{i \in HT} t_{exe_i} + \sum_{j \in XT} t_{exe_j} + \sum_{j \in XT} t_r + \sum_{j \in XT} t_{mem_j} + \sum_{\forall XputerCalls} t_{syn} - \sum_{j \in$$

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 timesbetween 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)). Memory re-mappings are necessary, when two Xputer tasks use the same data, but they are needing a different distribution of the data within the two-dimensional 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 (see figure 5) and has to be minimized for optimizing the acceleration-factor of an application (see equation (1)). The *COST*-function is determined each time from the profiler according to equation (1) using the actual viewed partitioning of the simulated annealing process:

Due to the data dependencies the profiler takes concurrent evaluations into consideration. The possible delay times for reconfiguring the Xputer during run time will be necessary, if a partition moves more basic blocks to the Xputer, than parallel GAG- or rather rALU-units are available, or if a fast on-chip reconfiguration is necessary for an Xputer-task. If possible, the reconfiguration is performed in parallel to running host tasks. For faster access all delay times are stored in a lookup table. For speed improvement of the simulated annealing algorithm, the cost increase or decrease due to the exchange is computed only. The rest of the tasks is not considered.

Simulated Annealing. The iterative partitioning phase is based on a simulated annealing algorithm [33]. Our algorithm computes different partitionings of Xputer tasks by varying their task allocation between host and Xputers. As above mentioned the initial partitioning is a resource-con-

straint list-based scheduling [6] 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.

Figure 5. The Simulated Annealing Algorithm

```
algorithm SIMULATED ANNEALING
begin
  temperature = INITIAL_TEMP.
  partitioning = INITIAL_PARTITIONING
  while (temperature > FINAL_TEMP. ) do
    for trials = 0 to MAXIMUM_TRIALS do
      new_partitioning = PERTURB(partitioning);
       C = COST(new_partitioning) - COST(partitioning);
      if (( C < 0) || (RANDOM(0,1) < exp(- C/T)))
      thenpartitioning = new_partitioning;
  temperature = SCHEDULE(temperature)
end.
```

The exchange function PERTURB (figure 5) uses several possibilities to change the partitioning randomly:

- A large Xputer task is split into several smaller tasks, and vice versa.
- One or more Xputer tasks are moved from one partition to the other.

The *PERTURB*-function is randomly choosing and using one of these two possibilities, because simulated annealing is a probabilistic algorithm of optimizing combinatorial problems, where the exchanging of elements is completely arbitrary.

For controlling the simulated annealing process, the *COST*-function can additionally be multiplied with a specific weight-function (e.g. exponential function), which results in a faster move to optimized solutions. The cooling schedule is controlled by a linear function $f(temp) = temp * 0.95$. The *RANDOM()*-function results a random number in the range between the first and second argument.

3.2 Experienced User Features

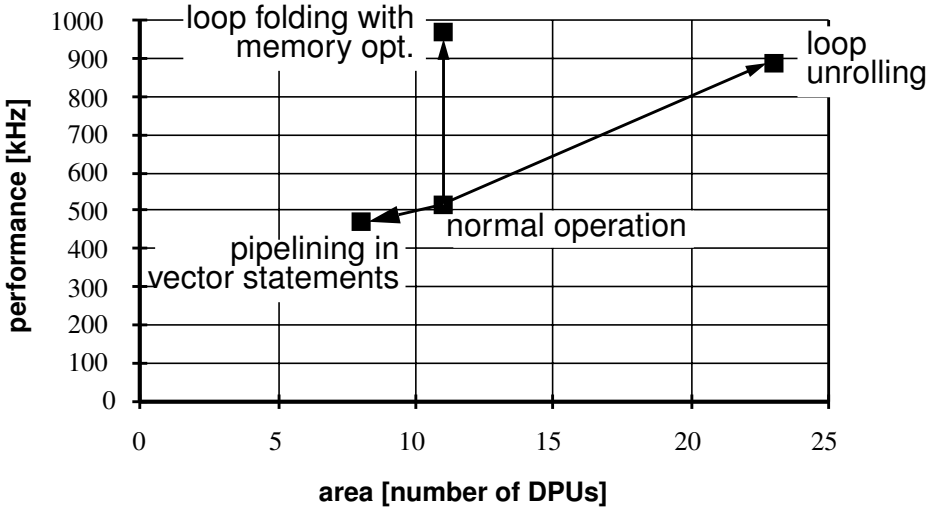
The experienced user can use and build optionally a generic function library with a large repertory of Xputer-library functions. The scan patterns as well as the rALU compound operators can be described in the language MoPl [1], which fully supports all features of the Xputer modules. The input specification 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-functions. Since X-C is an extension of ANSI C, also programming directly MoPL-parts into C-programs is possible, if no suitable library function is available. This optional data-procedural language features make highest acceleration factors provided by Xputer hardware possible.

3.3 Resource-driven second level Partitioning

Input of the X-C compiler in the second partitioning level of CoDe-X is a subset of ANSI C, which lacks dynamic structures. Such structures like pointers, operating system calls, etc. require a dynamic memory management to be run on the Xputer. Since the host's operating system takes care of memory management and I/O, the software parts for execution on the Xputer do not need such constructs. Especially for scientific computing, the restrictions of the C subset are not that important, since FORTRAN 77 lacks the same features and is most popular in scientific computing. Output of the X-C compiler is the organization of the datamap, the configuration code and parameter sets for the data sequencer, and the configuration code for the reconfigurable ALU. The datamap describes the storage scheme which determines the location of the variables in the memory. It is the main part of the interface between host and Xputer.

The X-C compiler translates input Xputer tasks into code which can be executed on the Xputer without further user interaction. It comprises four major parts: the data sequencer parameter generator (DSPG), the datapath synthesis system (DPSS), and the analysis/profiling tool (A/P) together with the partitioner. First the compiler performs a data and control flow analysis [18] [34]. The data structure obtained allows restructuring to perform parallelism like those done by compilers for supercomputers (e.g. vectorization of statements) or other optimization techniques like loop folding or loop unrolling [30], [31]. The data sequencer parameter generator maps the program's data in a regular way onto the two-dimensionally organized Xputer datamap, followed by a computation of the right address accesses (data sequencing) for each variable. A parameter file for the X-C compiler gives the current restrictions of the hardware, e.g. number of GAGs available, size of rALU subnets etc. This allows a flexible handling of the compiler for future up-grades of the accelerator hardware. The datapath synthesis system is responsible for the synthesis of the GAGs and rALU towards the rDPA. Due to a general interface between the rest of the X-C compiler and the synthesis system, it can be replaced easily by another synthesis tool, if required. The configuration of the rDPA is carried out in the following four phases: logic optimization and technology mapping, placement and routing, data scheduling, and finally the code generation. Partitioning of the operations onto the different rDPA chips is not necessary since the array of rDPA chips appears as one array of DPUs with transparent chip boundaries. For each Xputer task the X-C compiler produces a trade-off between area, that means number of DPUs in the rALU, and performance is computed and displayed in a graph. A small example is shown in figure 6

Figure 6. Example of performance/area trade-off graph of one Xputer task.



Now, for each Xputer task a single point in the trade-off graph is chosen. Since the size of the rDPA array is fixed, the point with the maximum performance which fits onto the rDPA of the Xputer module is taken. For this task the configuration set, the required time for configuration, and the execution time is stored. Usually the following optimizations are performed:

- fully nested loops: loop unrolling (or for large compound operators: loop folding)
- not fully nested loops:
  - inner loop: loop folding (or for small compound operators: loop unrolling)
  - outer loop: vectorization (to save area), or normal operation

The DPSS also gives the execution time of the rALU compound operator synthesized in the rDPA. Since the worst case delay time of the individual operators are known in advance, the data scheduling delivers the complete execution

time. Further details about the X-C compiler and the four phases of the DPSS, as well as examples of the mapping onto the rDPA can be found in [10] [19] and [20].

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.

Figure 7. Transforming for-loop by using strip mining with block size 1000

```
.for (row=0; row<6400-const1; row++)
. {
.   for (col.=0; col.<6400-const2+1; col.++)
.   {
.     sumval = Pix.[(row+0)*256 + (col.+0)] * Coeff[0];
.     sumval += Pix.[(row+0)*256 + (col.+1)] * Coeff[1];
.     sumval += Pix.[(row+0)*256 + (col.+2)] * Coeff[2];
      sumval += Pix.[(row+1)*256 + (col.+0)] * Coeff[3];
      sumval += Pix.[(row+1)*256 + (col.+1)] * Coeff[4];
      sumval += Pix.[(row+1)*256 + (col.+2)] * Coeff[5];
      sumval += Pix.[(row+2)*256 + (col.+0)] * Coeff[6];
      sumval += Pix.[(row+2)*256 + (col.+1)] * Coeff[7];
      sumval += Pix.[(row+2)*256 + (col.+2)] * Coeff[8];

      Pix._new[((row+dead_rows)*256+(col.+dead_cols))] = sumval/normal_factor;
    }
  }
```

Strip Mining

```
for (ind. = 0; ind.<6400-const1; ind.=index+1000)
{
 for (row=ind.; row<min(ind.+1000,6400-const1); row++)
. {
.   for (col.=0; col.<6400-const2+1); col.++)
.   {
.     sumval = Pix.[(row+0)*256 + (col.+0)] * Coeff[0];
.     sumval += Pix.[(row+0)*256 + (col.+1)] * Coeff[1];
```

o
o

Neighborhood averaging is a straightforward spatial-domain technique for image smoothing [7]. 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:

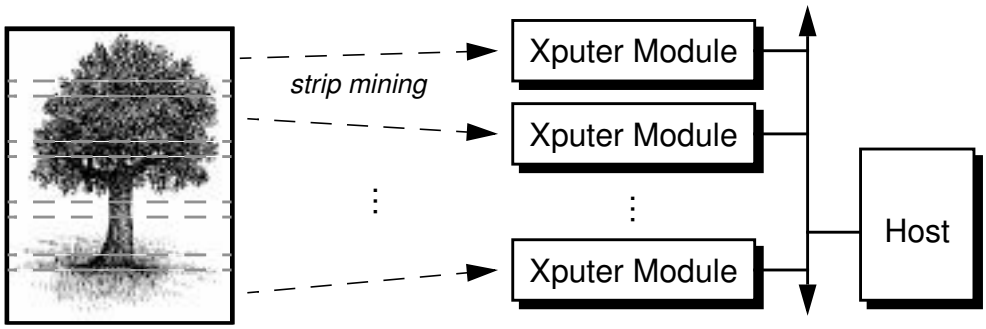
$$g(x, y) = \frac{1}{M} \sum_{(n, m) \in S} f(n, m) \qquad \text{(eq. 2)}$$

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, because one for-loop could be transformed according to figure 7.

The resulting smaller independent loops can executed in parallel on different Xputer modules (see figure 8). The X-C compiler was performing loop unrolling additionally up to the limit of available hardware resources (see section 3.3).

The profiler was computing the host- and Xputer- cost functions for them. The data dependencies require a sequential execution of the four tasks, which influenced the overall execution time of this application. The final decision was to shift the two candidates for the Xputer to the accelerator. The X-C compiler on the second level of hardware/software co-design was then mapping these basic blocks onto the Xputer. In this final solution the Xputer must be configured only one time. The address generation for the data accesses in the fully nested loops of the two Xputer tasks was handled each time by one generic address generator. The final acceleration factor in comparison with a SUN SPARC 10/51 was 73.

Figure 8. Image processing example distributed on different Xputer modules



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. One of the new features is the two level partitioning approach. An extension of C, including also data-procedural features, is accepted as input. The profiling-driven host/accelerator partitioning for performance optimization as well as the resource-parameter-driven sequential/structural partitioning for accelerator programming are carried out. At first level, performance critical parts of an algorithm are localized and shifted to the Xputer without manual interaction. The second level carries out a resource-driven compilation/synthesis from accelerator source code to optimize the utilization of its reconfigurable datapath resources.

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] T. Ball, J. R. Larus: Branch Prediction for free; Proc. of the ACM-SIGPLAN ´93: Conf. on Programming Language Design & Implement.ation, pp. 300 - 313; ACM-Press, 1993

[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] A. Despain, I.-J. Huang: Synthesis of Application Specific Instruction Sets"; IEEE-Trans on CAD of Integrated Circuits and Systems, Vol. 14, no. 6, June 1995

[5] R. Ernst, J. Henkel, T. Benner: Hardware-Software Cosynthesis for Microcontrollers; IEEE Design & Test, pp. 64-75, Dec. 1993

[6] D. D. Gajski, N. D. Dutt, A. C.-H. Wu, S. Y.-L. Lin: High-Level Synthesis, Introduction to Chip and System Design; Kluwer Academic Publishers, Boston, Dordrecht, London, 1992

[7] Gonzalez R. C., Wintz P.: *Digital Image Processing;* Addison-Wesley Publishing Company, USA 1977

[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] 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, March 1995

[10] R. W. Hartenstein, R. Kress: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; Asia and South Pacific Design Automation Conference, ASP-DAC'95, Nippon Convention Center, Makuhari, Chiba, Japan, Aug. 29 - Sept. 1, 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, Smolenice, Slovakia, Sept. 1995

[14] R. W. Hartenstein, H. Reinig: Novel Sequencer Hardware for High-Speed Signal Processing; Workshop on Design Methodologies for Microelectronics, Smolenice Castle, Slovakia, Sept. 11-13, 1995

[15] Reiner W. Hartenstein, Jürgen Becker, Michael Herz, Rainer Kress, Ulrich Nageldinger: A Partitioning Programming Environment for a Novel Parallel Architecture; 10th International Parallel Processing Symposium (IPPS), Honolulu, Hawaii, April 1996

[16] R. Hartenstein, J. Becker, R. Kress: Custom Computing Machines vs. Hardware/Software Co-Design: from a globalized point of view; submitted for FPL'96, Darmstadt, 1996

[17] T. A. Kean: Configurable Logic: A Dynamically Programmable Cellular Architecture and its VLSI Implementation; Ph.D. Thesis, University of Edinburgh, Dept. of Computer Science, 1989

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

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

[20] R. Kress: Computing with reconfigurable ALU arrays; IT Press (planned for 1996)

[21] 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

[22] 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

[23] 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

[24] P. Marwedel, G. Goossens (ed.): "Code Generation for Embedded Processors", Kluwer Acad. Publishers, 1995

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

[26] K. L. Pocek, D. Buell: Proc. IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'93, Napa, CA, April 1993

[27] see [26], but 1994, 1995, and 1996

[28] 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

[29] Sato et al.: An Integrated Design Environment for Application Specific Integrated Processors, Proc. ICCD 1991

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

[31] K. Schmidt: Xputers: a high performance computing paradigm; IT Press (planned for 1996)

[32] Z. Shen, Z. Li, P.-C. Yew: "An Empirical Study of Fortran Programs for Parallelizing Compiler"; *IEEE Transactions on Parallel and Distributed Systems*, Vol.1, No.3, pp. 356-364, July 1990.

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

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

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

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