Reiner W. Hartenstein, Jürgen Becker, et al.: Two-Level Hardware/Software Partitioning Using CoDe-X; Int. IEEE Symposium on Engineering of Computer Based Systems (ECBS), Friedrichshafen, Germany, March 1996

Two-Level Hardware/Software Partitioning Using CoDe-X

Reiner W. Hartenstein, Jürgen Becker, Rainer Kress

University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, e-mail: abakus@informatik.uni-kl.de

Abstract

The paper presents a two level hardware/software partitioning strategy of a co-design framework (CoDe-X), using an Xputer as accelerator with reconfigurable datapath. CoDe-X accepts C-programs and carries out both, profiling-driven host/accelerator partitioning and (2nd level) resource-parameter-driven sequential/structural partitioning of the accelerator source code to optimize the utilization of its reconfigurable datapath resources.

1. Introduction

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], [10], [23]. Further co-design approaches are advocated explicitly or implicitly in developing custom computing machines (CCMs: [16], [26]) such within the R&D scenes of ASIPs [5] [25] [29] and FCCMs [7] [8]. 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) [13] or the rDPA approach (reconfigurable data path array) by Rainer Kress [19] [20].

For such a new class of hardware platforms a new class of compilers is needed, which generate both, sequential and structural code: partitioning compilers, which partition a source into cooperating structural and sequential code segments. In such an environment hardware/software co-design efforts require two levels of partitioning: host/accelerator partitioning (first level) for optimizing performance and a structural/sequential partitioning (second level) for optimizing the hardware/software trade-off of the Xputer resources. This paper presents a framework based on this paradigm. The hardware/software co-design framework CoDe-X targets the partitioning and compilation of conventional C programs onto a host using the Xputer as universal configurable accelerator. In contrast to other hardware/software co-design approaches no additional programming extensions are required and the framework use is not limited to hardware experts. The partitioning is based on a simulated annealing algorithm. 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 image processing.

2. The Target Hardware

The target hardware consists of a host workstation that uses the Xputer as hardware accelerator (see Figure 1). The key concept of an Xputer [11] [13] [14], is to map the structure of performance critical algorithms onto the hardware architecture. Performance critical algorithms typically iterate the same set of operations over a large amount of data. Since ever the same operations are applied, an Xputer has a reconfigurable parallel arithmetic-logic unit (rALU), which can implement complex operations on multiple input words and compute multiple results. All input and output data to the complex rALU operations is stored in a so-called scan window (SW). A scan window is a programming model of a sliding window, which moves across data memory space under control of a data sequencer. All data in the scan windows can be accessed in parallel by the rALU.

The large amount of input data typically requires an algorithm to be organized in (nested) loops, where different array elements are referenced as operands to the computations of the current iteration. The resulting sequence of data accesses shows a regularity, which allows to describe such sequences generically from a few parameters. An Xputer data sequencer (DS) provides seven hardwired generic address generators (GAGs), which are capable of interpreting such parameter sets to compute a generic data address sequence (SP: scan pattern). This address computation does not require memory cycles. That’s why it provides substantial speed-up.

To be run on an Xputer, an algorithm has to be transformed into parameters sets controlling the generic

address generators and a configuration of the rALU, which implements the data manipulations within a loop body. Data manipulations are transport-triggered. I. e., each time, the GAG places the SW to a new memory location, the complex operations configured into the rALU are evoked automatically. Since locality of data is essential in finding a generic SP, a compiler for an Xputer first has to create a good and regular storage scheme (DM: data map, see Figure 2.) to obtain good results in performance optimization.

Our current prototype of the Xputer consists of seven Xputer modules (X-Module) and operates as a kind of configurable co-processor to a host computer (see Figure 1). Each X-Module consists of a data sequencer (DS), a rALU subnet and at least two megabytes of local memory. All X-Modules can operate in parallel. Apart from the local memory access, two means of communication between X-Modules are available. First, the rALU subnets can exchange internal results with their neighbours without disturbing parallel activity. Second, the Data Sequencers can access data memory and rALU subnets on other X-Modules using the global bus. But this can be done only sequentially. The global bus is used by the Xputer controller (CTRL) as well to reconfigure the data sequencers and the rALU, whenever necessary. This controller is the coordinating instance of the Xputer, which ensures a well-defined parallel activity of the data sequencers by selecting the appropriate parameter sets for configuration.

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 memory on the Xputer via the bus interface, and on the other hand the Xputer can access data in the host's memory, so that global communication is

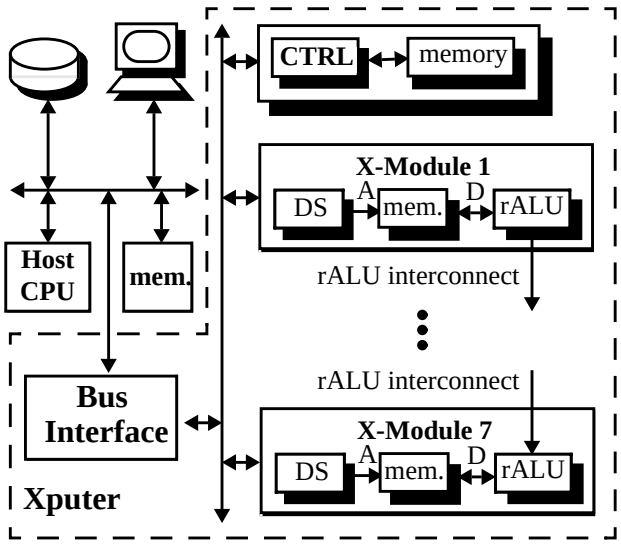

Figure 1. The underlying hardware platform

available. The reconfigurable datapath architecture (rDPA) which serves as rALU has been developed especially for computing statement blocks from loops as well as other arithmetic and logic computations [12].

The only connection to the host computer is through the bus interface. That way, the Xputer could be interfaced to a different host with minimal effort. The part of the bus interface that is inserted into the host's backplane would have to be redesigned according to the host's bus specifications. All other parts could remain the same.

3. Parameter-driven H/S Partitioning

This section gives an overview on the two-level hardware/software co-design framework CoDe-X. Input language to the framework is the programming language C. The input is partitioned in a first level into a part for execution on the host

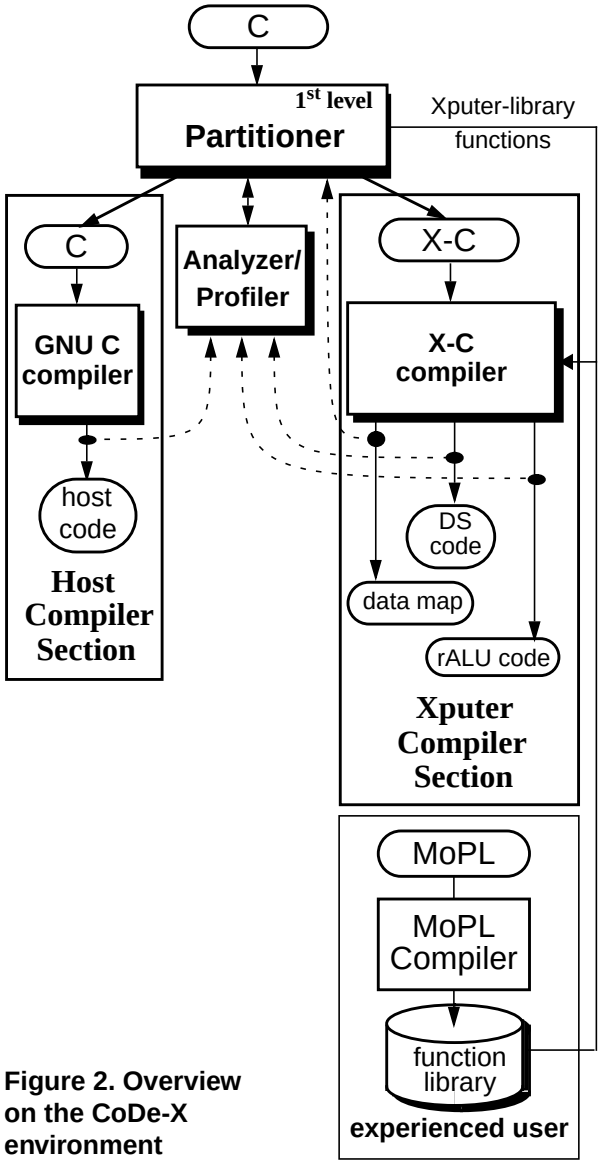

Figure 2. Overview on the CoDe-X environment

and a part for execution on the Xputer. Possible parts for the execution on the Xputer may be described in Xputer-C (X-C). X-C is an extended subset of the programming language C, which lacks only dynamic structures, but includes language features to express scan patterns. The X-C input is then partitioned in the second level by the X-C compiler in a sequential part for programming the data sequencer, and a structural part for configuring the rALU. Special Xputer-library function-calls in the original C description of the applications allow to take full advantage of the high acceleration factors possible by the Xputer paradigm. Experienced users may add new functions to the library such that each user can access them (see Figure 2).

First the host/Xputer partitioning is explained (next section). Then the programming of the Xputer is shown.and finally the options for experienced users are discussed.

3.1 Profiling-driven 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 and are filtered out in a first step:

- host tasks,
- Xputer tasks, and
- Xputer-library functions.

The host tasks have to be computed on the host, since they contain dynamic structures or call operating system utilities, which 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.

At the beginning, a filtered input program for the Xputer is represented as a graph, called flow graph. This graph is divided into several Xputer tasks. An Xputer task can be computed on a single Xputer module and is defined as follows:

- 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).

This means a basic block, which is a linear sequence of statements, can be included in several Xputer tasks (figure 4). The restriction of seven loops for one Xputer task is caused by the current Xputer prototype. The data sequencer of this prototype contains seven GAGs, which can operate exclusively. The resulting task graph has several alternative paths in such a case. A data dependency analysis based on the GCD-test [34] gives the data dependencies between the tasks. 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 a 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 (see Figure 3). This transformation is called strip mining [22].

This makes it possible to compute these chunks in parallel on different Xputer modules. In our approach the block size of the chunks depends on parameters describing the hardware resources of the current Xputer prototype in order to achieve an optimized performance/area trade-off. This technique can be used e.g. in image processing applications (see Section 4), if an image is divided in stripes of equal sizes in order to manipulate these stripes concurrently. The optimization techniques [22] 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 which fits onto an Xputer module is chosen. This 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 may be two problems: if the accelerator needs reconfiguration at run-time and in a few cases, if the datamap has to be reorganized also at run-time.

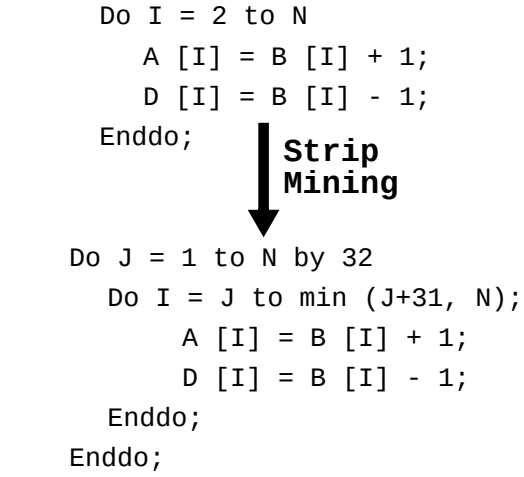

Figure 3. Strip mining example (block size: 32)

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 an 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 idling 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. Currently only the FPGAs from Algotronix [17] (the new Xilinx XC6200 series [4]), Atmel [2], and from National Semiconductors [27] support partial configuration.

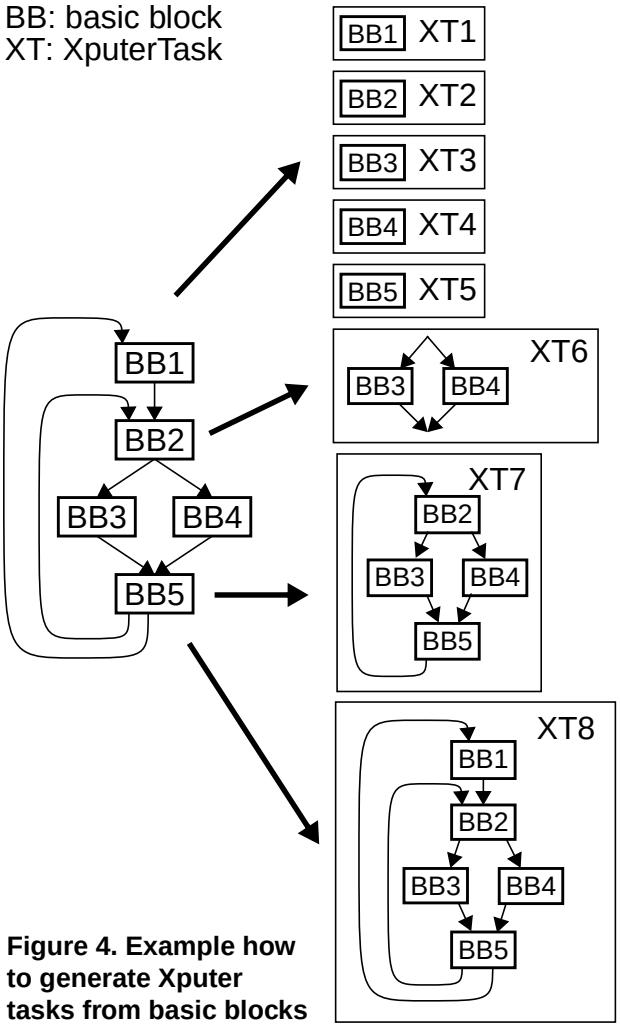

Figure 4. Example how to generate Xputer tasks from basic blocks

- 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 96 datapath units.

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.

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 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 [3] are considered by computing the execution time of a code segment. The techniques used are similar to these from Malik et. al. [24].

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)

$$\sum_{j \in XT} t_{mem_j} + \sum_{\forall XputerCalls} t_{syn} - \sum_{j \in XT} t_{ov}$$

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

$$t_{acc} = \sum_{i \in HT} t_{exe_i} + \sum_{j \in XT} t_{exe_j} + \sum_{j \in XT} t_r +$$

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 basic blocks (host and Xputer), plus the above mentioned penalty times (see t_{acc}, equation (1)). Memory re-mappings are necessary, when two basic blocks use the same data onto the Xputer, 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 basic block 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 2) 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 basic blocks are not considered.

Simulated Annealing. The iterative partitioning phase is based on a simulated annealing algorithm [20]. Our algorithm computes different partitionings of Xputer tasks by varying their task allocation between host and Xputer.s. The Xputer tasks of the initial partitioning are swapped. If the new partitioning results **i**n 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.

```
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)))
        then      partitioning = new_partitioning;
  temperature = SCHEDULE(temperature)
end.
```

Figure 5. The Simulated Annealing Algorithm

The exchange function PERTURB (see figure 2) 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.

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.

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 Resource-driven 2nd level Partitioning

The X-C compiler realizes the 2nd level of partitioning of CoDe-X and translates an X-C program 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.

The partitioner at 2nd level (see Figure 2.) performs a data and control flow analysis. First the control flow of the program graph is partitioned according to the algorithm of Tarjan [33] 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 have not been considered until now. So second the obtained coarse-grained components have to be transformed such that:

- the resulting sequence comprises blocks of possibly similar size, and
- a good exploitation of the given hardware resources can be guaranteed by using available parallelism

This second partitioning of the data flow is improved by using a heuristic, which control the computational steps exploiting the special Xputer hardware resources in an optimized way [30]. This heuristic analyzes, if the structure of statement sequences can be mapped well to the Xputer hardware avoiding reconfigurations or idle hardware resources. Xputers provide best parallelism at statement or expression level. So in our approach we try to vectorize the statement blocks in nested for-loops according to the vectorization algorithm of J. R. Allen and K. Kennedy [18], after a data dependency analysis has been performed [34].

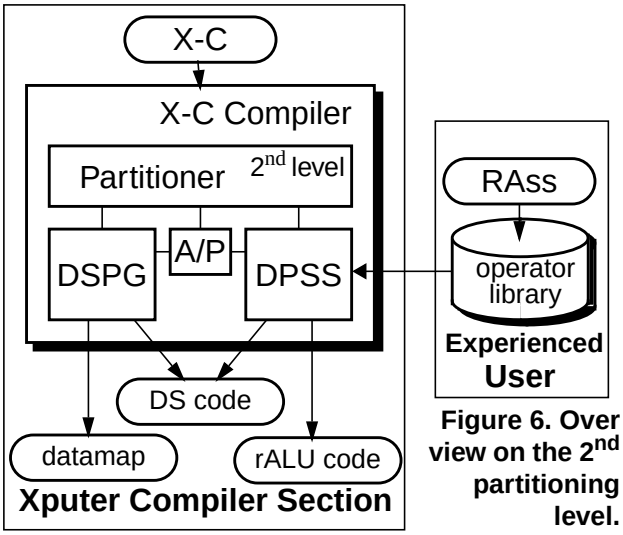

Figure 6. Over view on the 2nd partitioning level.

The data sequencer parameter generator (DSPG) maps the programs 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 (see Figure 2.). This tool is computing the access sequences of each variable used inside of nested for loops according to the linear index functions of these variables. This means that the for loops of X-C programs are mapped onto parameters for configuring one generic address generator of the data sequencer, which is then generating the corresponding physical addresses only by hardware.

The access sequences of up to 4 fully nested loops can be computed by this tool. 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. This allows a flexible handling of the compiler for future upgrades of the Xputer hardware. The datapath synthesis system is responsible for the synthesis towards the rALU. The code optimizations of loop folding (pipelining across iterations of for loops) and loop unrolling are performed by the DPSS whenever possible. In the current version the rDPA serves as rALU. 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 task of configuring 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 statements onto the different rDPA chips is not necessary since the array of rDPA chips appears as one array with transparent chip boundaries. The DPSS also gives the execution time of the compound operator in the rDPA. Since the worst case delay times 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 [30] and [12].

For each Xputer task the X-C compiler produces a trade-off between area, that means amount of rDPA resources, and performance is computed and displayed in a graph.

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 by the X-C compiler: ●

- 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 selection of the possible implementation of an Xputer task is fully automatic and requires no user interaction.

3.3 Experienced Users

The experienced user is responsible for building 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 C-program (see figure 2). The rALU assembler (RAss) allows the extension of the operator library which is used as a base for the configuration of the rDPA (see figure 2). All operators of the programming language C are in the operator library by default. User specific functions for the datapath units can be added.

```
.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];
```

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

4. Example: Smoothing 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 [9]. 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.

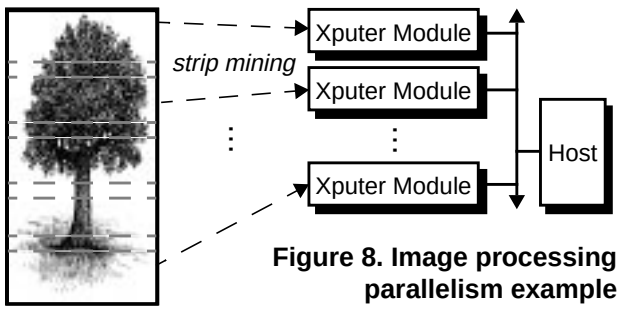

Figure 8. Image processing parallelism example

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 3.2).

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.

5. Conclusions

CoDe-X, a two-level partitioning hardware/software co-design framework 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 co-design approach. CoDe-X accepts C-programs and carries out the profiling-driven host/accelerator partitioning for performance optimization and resource-parameter-driven sequential/structural partitioning for accelerator programming. 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. From several application examples encouraging acceleration factors have been obtained in comparison to a single workstation without an accelerator-board.

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] N. N.: Configurable Logic, Design and Application Book; Atmel Coporation, San Jose, CA, 1994

[3] 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

[4] S. Churcher, T. Kean, B. Wilkie: The XC6200 FastMap™ Processor Interface; 5th Int. Workshop on Field-Programmable Logic and Applications, FPL'95, Oxford, UK, Aug./Sept. 1995

[5] 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

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

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

[8] see [7], but 1994, 1995, and 1996

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

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

[11] 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

[12] 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

[13] 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;

[14] see [13]: also in: Future Generation Computer Systems no. 7, pp. 181-198 (North-Holland, 1991/92)

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

[16] R. Hartenstein, J. Becker, R. Kress: Customized Computers: a generalized survey; 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] L. Lin: High-Level Synthesis, Introduction to Chip and System Design; Kluwer Acad. Publ., Boston, London, 1992

[22] 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

[23] 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

[24] 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

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

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

[27] N. N.: Configurable Logic Array (CLAy™); Preliminary Datasheet, National Semiconductors, Santa Clara, CA, Dec. 1993

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

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

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