Reiner W. Hartenstein, Jürgen Becker, et al.: A Novel Hardware/Software Co-Design Framework; Journal of the Brazilian Computer Society: Special Issue on Electronic Design Automation, no.2, vol. 2, pp. 16-26, November 1995

A Novel Hardware/Software Co-Design Framework

Reiner W. Hartenstein, Jürgen Becker, Rainer Kress, Helmut Reinig

University of Kaiserslautern

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

Abstract

A novel hardware/software co-design framework (CoDe-X) is presented in this paper, where an Xputer is used as universal accelerator based on a reconfigurable datapath hardware. CoDe-X accepts C-programs and carries out both, the profiling-driven host/accelerator partitioning for performance optimization, and the resource-driven sequential/structural partitioning of the accelerator source code to optimize the utilization of its reconfigurable datapath resources.

1. A Comparative Introduction

The scene of hardware/software co-design has introduced a number approaches to speed-up performance, to minimize hardware/software trade-off and to reduce total design time.

Ernst and Henkel [EHB93] use a system for hardware/software co-design of embedded controllers. Input of the system is the programming language C with additional real-time programming extensions. The partitioning process starts with a complete software solution. The parts which violate timing constraints are moved into code segments for implementation onto hardware. These violations are detected by simulation instead of code analysis since a partitioning produces several timing overheads. For the partitioning a simulated annealing algorithm is used.

The complementary approach has been adapted by Gupta and DeMicheli [GCM94] from Stanford University. Systems are specified by using the hardware description language Hardware-C and the non-critical functions are then shifted from the hardware to software programs running on a conventional processor. Hardware cost is reduced without sacrificing performance, but currently using the framework is limited to hardware experts only.

Luk, Lu and Page [LWP94] described a method for partitioning functional programs, which operate on multidimensional arrays, by delegating computation-intensive operations to an application-specific hardware co-processor. The system partitions the program into three steps: “divide”, “conquer” and “merge”. A general-purpose processor, usually efficient in executing relatively complex, data-dependent and variable size operations, can be used for dividing the input data into smaller pieces. Purpose-built hardware is used for processing the partitioned data, often in systolic mode.

Further co-design approaches are advocated by the FCCM-scene [FCCM94], where a von Neumann host implementation is extended by adding external operators, which are configured on field-programmable circuits. Usually this configuration requires hardware experts. The von Neumann paradigm does not support efficiently “soft” hardware: because of its extremely tight coupling between instruction sequencer and ALU. So a new paradigm is required like the one of Xputers, which con-

veniently supports “soft” ALUs like the rALU concept (reconfigurable ALU). 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 for optimizing performance and a structural/sequential partitioning 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 the approach of Ernst and Henkel no additional programming extensions are required. The partitioning is based also on a simulated annealing algorithm. Divide and conquer strategies used from Luk, Lu and Page are good examples for an efficient partitioning into parts for the host and the accelerator. The approach described in this paper is not limited to such algorithms. The fundamentally new feature of the CoDe-X framework is the two-level co-design approach.

First this paper describes the underlying hardware. Section 3 presents the hardware/software co-design framework CoDe-X. The different algorithms for the partitioning are shown and the used tools for evaluating the compilation and synthesis task are sketched.

2. The Underlying Hardware

The underlying hardware consists of a host workstation that uses the Xputer as hardware accelerator. The key concept of an Xputer [AHR94], [HHR91] is to map the structure of performance critical algorithms onto the hardware architecture. Performance critical algorithms typically apply the same set of operations to a large amount of data. Since ever the same operations are applied, an Xputer has a reconfigurable arithmetic-logic unit (rALU), which can implement complex operations on multiple input words and compute multiple results (figure 1). 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 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 this sequence by a number of parameters. An Xputer data sequencer provides seven hardwired generic address generators (GAGs), which are capable of interpreting such parameter sets to compute generic address sequences for the scan windows. To be run on an Xputer, an algorithm has to be transformed into parameters sets for the generic address generators and a configuration of the rALU, which implements the data manipulations within a loop body. Each time, the generic address generators have accessed all input data of a loop iteration, the complex operations configured into the rALU are applied to the input data and the outputs can be written to the data memory by the generic address generators. Since a compiler for an Xputer may rearrange the data in memory to optimize the data access sequences, a data map is required to describe the distribution of input and output data in the data memory.

The current prototype of the Xputer operates as a kind of configurable co-processor to a host computer. All I/O operations are done by the host as well as the memory management. The host has direct access to all memory on the Xputer, and on the other hand the Xputer can access data in the host’s memory. The reconfigurable datapath architecture (rDPA) which serves as rALU has been developed especially for evaluating statement blocks in loops and other arithmetic and logic computations [HKR94]. The rDPA consists of a regular array of identical processing elements called datapath units (DPUs, figure 2). Each DPU has two input and two output registers. The dataflow direction is from west and/or north to east and/or south. The operation of the DPUs is data-driven, the operation is performed as soon as the required operands are available. The communication between the

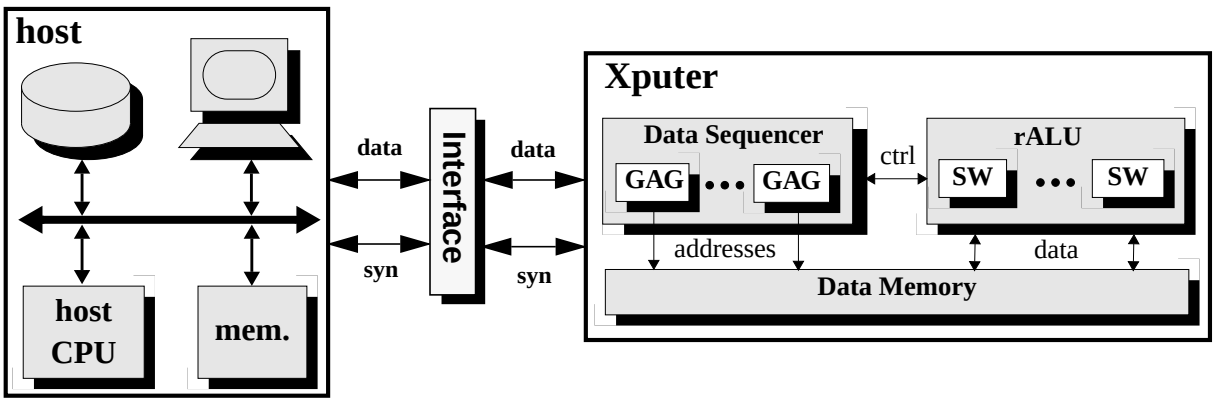

Figure 1. Block Diagram of the underlying hardware

neighbouring DPUs is synchronized by a handshake. A global I/O bus has been integrated into the rDPA, permitting the DPUs to write from their output registers directly outside the array and to read directly from outside. This means, that input data to expressions mapped into the rDPA does not need

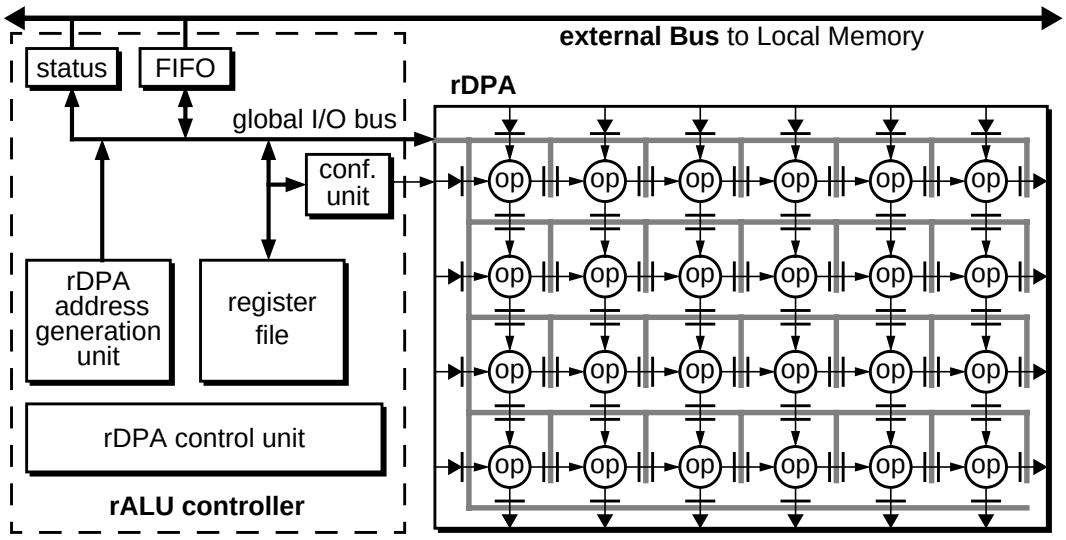

Figure 2. rALU consisting of a reconfigurable datapath architecture with rALU controller

to be routed through the DPUs. The communication between an external controller, or host, and the DPUs is synchronized by a handshake like the internal communications. An extensible set of operators for each DPU is provided by a library. The set includes the operators of the programming language C. The operators of the DPUs are configurable. A DPU is implemented using a fixed ALU and a microprogrammed control. The configuration is data-driven, and therefore special timing does not have to be considered. Together with the rALU controller the rDPA forms the data-driven rALU, as shown in figure 2. The control chip consists of a control unit, a register file, and an address generation unit for addressing the DPUs. The register file is useful for optimizing memory cycles. The rDPA control unit holds a program to control the different parts of the data-driven rALU. An earlier version of the rDPA is described in [HKR94].

3. The Hardware/Software Co-Design Framework CoDe-X

This section gives an overview on the 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 and a part for execution on the accelerator. In this case the Xputer is used as accelerator. Possible parts for the execution on the Xputer are described in Xputer-C (X-C). X-C is an almost complete subset of the programming language C, which lacks only dynamic structures. The X-C input is then partitioned in a second level in sequential part for programming the data sequencer and a structural part for configuring the rALU. This rALU description is used for further synthesis in the datapath synthesis system (DPSS). The output of the X-C compiler results in three files: a rALU file for the configuration of the rALU, a parameter set for the loading of the data sequencer (DS code), and a description of the datamap (IF descr.). The description of the datamap gives the location of the variables in the memory. Thus it describes a part of the interface between the host and the Xputer. The part of the C program which is partitioned onto the host is compiled with the GNU C compiler. Figure 3 gives a detailed overview on the hardware/software co-design framework CoDe-X.

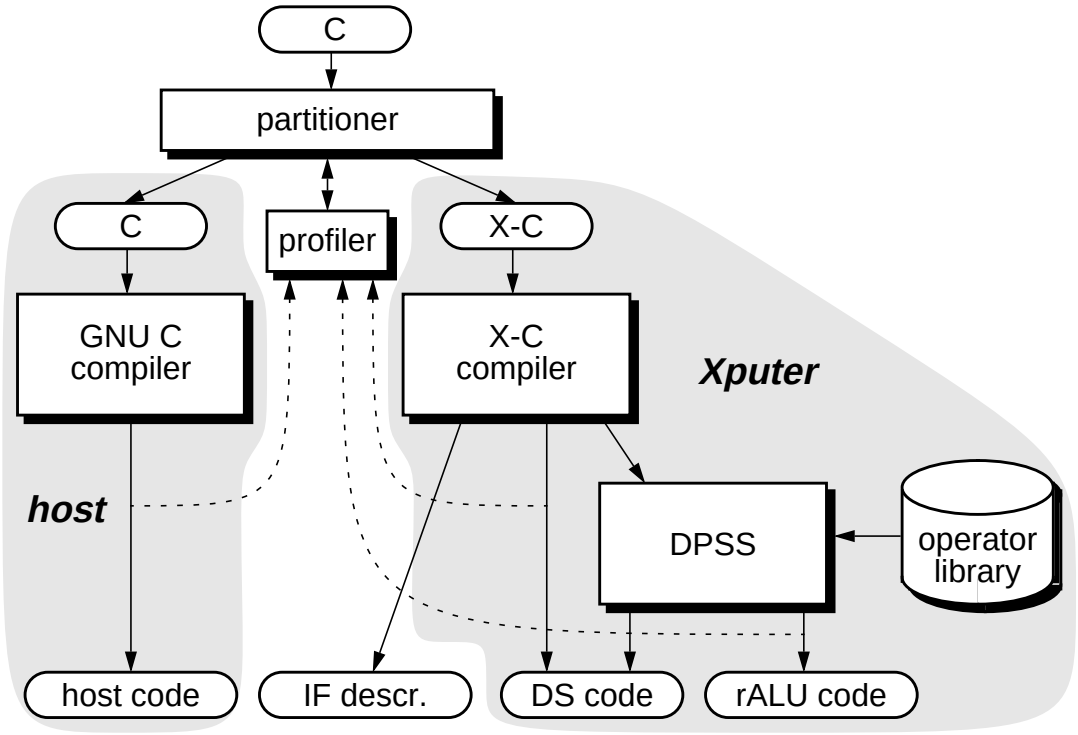

Figure 3. Overview on the CoDe-X environment

3.1 Partitioning based on Profiling

The partitioner in the first level generates three-address statements of the input C program which are then represented as a graph, called flow graph. This flow graph is partitioned into basic blocks. A basic block is a sequence of consecutive statements in which the flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end. Figure 4a shows an example of a graph representation of the basic blocks with their control flow. A data dependency analysis based on the GCD-test [WoTs92] gives the data dependencies between the basic blocks (figure 4b). In this phase some heuristics can be introduced that reduce the number of basic blocks.

Basic blocks in nested loops may be merged together to a single basic block, if it fits onto the reconfigurable datapath architecture. This heuristic is based on the knowledge that nested loops achieve high performance factors on Xputers. The area required can be approximated by the number of operators in the statement blocks of these loops. Rearranging the basic blocks according to their data dependencies results in a data flow graph where the basic blocks are represented by the nodes (figure 4c). The basic blocks that can be evaluated concurrently are displayed side by side.

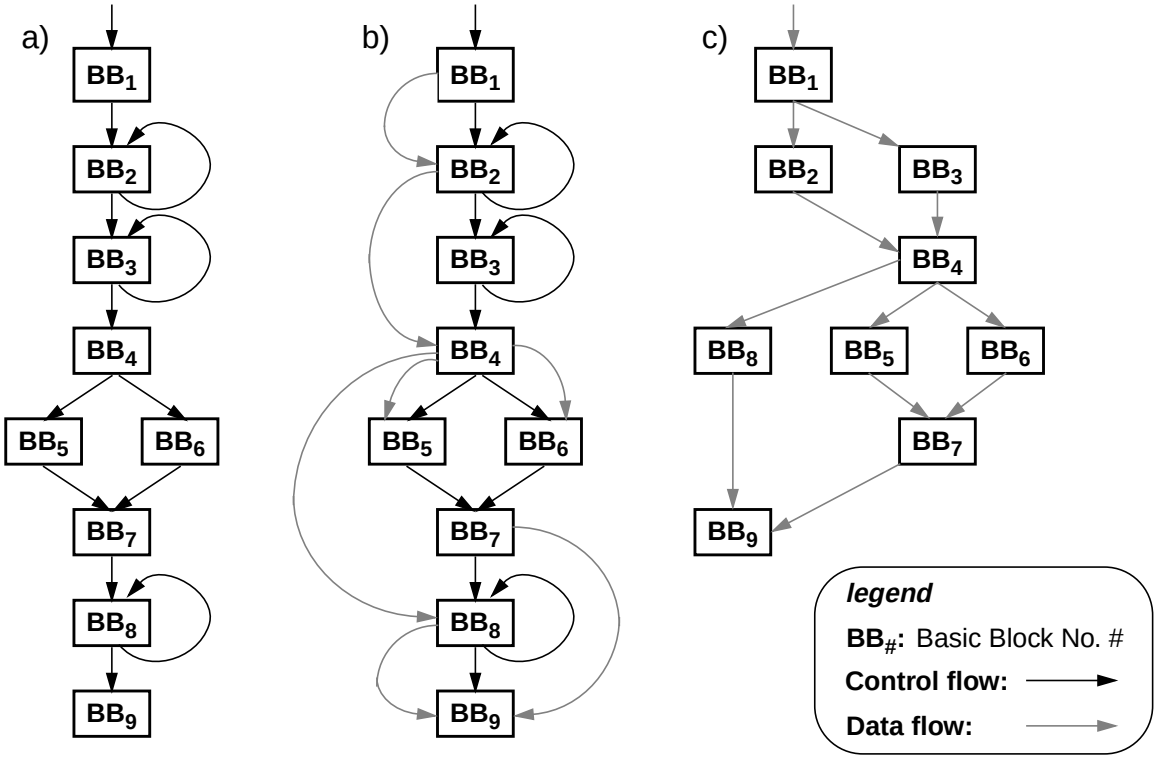

Figure 4. Graph representation of the basic blocks: a) control flow only, b) with data dependencies, c) data flow only showing possible parallelism between basic blocks

Profiler. The partitioning into parts for the host and parts for the Xputer is based on the performance analysis of a profiler. For each basic block the time for evaluation on the Xputer as well as the evaluation on the host is approximately computed. The performance data for the Xputer task is received from the datapath synthesis system (DPSS) and the X-C compiler. The X-C compiler is able to evaluate the number of iterations for the statement blocks. Knowing the evaluation time for the statement block from the datapath synthesis system, the complete evaluation time can be approximated. The time consuming placement of the DPSS does not have to be performed for approximating the execution time of the statement blocks.

The optimizations provided by the X-C compiler and datapath synthesis system allow to realize several implementations for the reconfigurable datapath architecture with different performance/area trade-offs in the second level of the partitioning of CoDe-X. This trade-off is called design space. Figure 5 marks the regions that can be reached with the different optimization techniques. Pipelining in vector statements can be used for all implementations where vector statements occur. It reduces the area of the implementation drastically. When the realized implementation is bound by I/O operations this may be done without loosing performance. In other cases the speed will decrease slightly. Loop folding and loop unrolling can be used when the statement block occurs in inner loops only. Loop folding uses pipelining over loop iterations or loop boundaries. This technique will always increase the performance without requiring additional area. Thus this method is very attractive and used by default by the DPSS. Loop unrolling results always the highest speed. Depending on the number of unrolled iterations, the area increase may be significant. For data dependencies between

the iterations of the loop (overlapping scan windows) memory cycle optimization can be applied resulting in a further increase of speed. Of course the proposed methods can be combined to reach further positions in the design space.

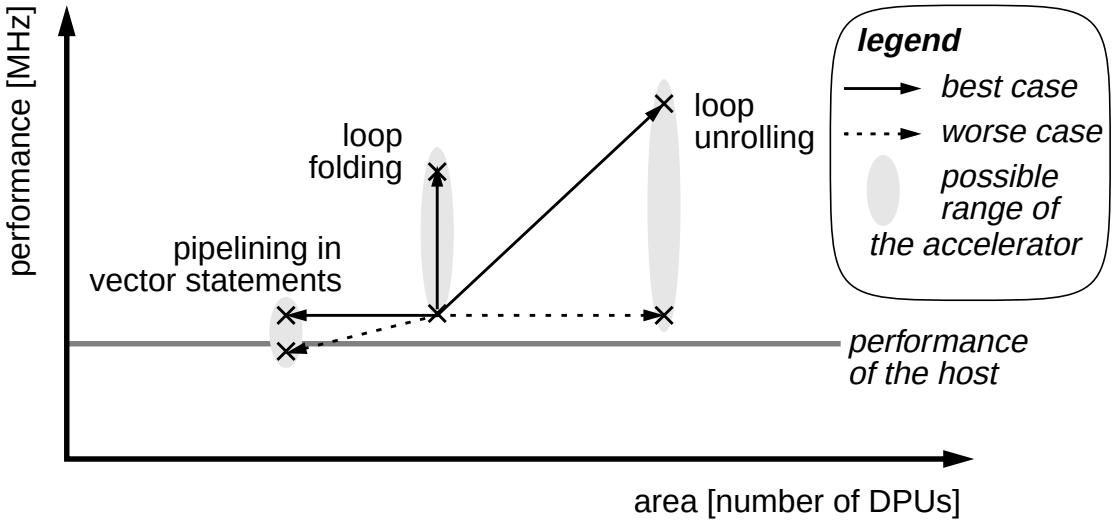

Figure 5. Example of a performance/area trade-off of the basic blocks

The performance of the host is approximated by examining the assembler code. For each processor type and workstation another model is required. The behaviour of the underlying hardware and operating system has to be deterministic and known. This implies the timing behaviour 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 behaviour, and no asynchronous interrupts should be allowed [PuKo89]. Figure 5 shows a grey shaded line giving the performance of the host. For each basic block several acceleration factors can be computed. For each point in the design space that can be reached with the Xputer implementation the acceleration factor is computed as:

$$\text{acceleration factor} = \frac{\text{performance Xputer}}{\text{performance host}} \quad (1)$$

Simulated-Annealing. The partitioning phase is based on a simulated annealing algorithm [Sher93]. This algorithm uses iterative improvement and it belongs to the probabilistic algorithms. The basic blocks of a chosen 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 which depends on the temperature of the annealing process. This avoids that the algorithm gets stuck in a local minimum. The temperature is very high at the beginning and is gradually lowered.

In general the costs are determined by delay times of the basic blocks, plus the penalty cost for the synchronization. Figure 6 shows the algorithm for simulated annealing partitioning.

First, an initial partitioning for the simulated annealing process is computed by placing the basic blocks in two partitions based on their acceleration factor. One partition belongs to basic blocks that are executed on the host, the other partition belongs to basic blocks that are executed on the Xputer (figure 7). Since the X-C compiler lacks the possibility to compile dynamic structures, these structures including all I/O routines are always executed on the host.

```
algorithm SIMULATED ANNEALING
begin
    temperature = INITIAL_TEMPERATURE
    partitioning = INITAL_PARTITIONING
    while (temperature > FINAL_TEMPERATURE ) 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 6. The simulated annealing algorithm

The *PERTURB*-function uses several possibilities to change the partitioning randomly:

- A basic block for evaluation on the host can be changed with a basic block for evaluation on the Xputer.
- A basic block can be moved into the other partition.
- For basic blocks that occur in inner loops on the Xputer, loop unrolling instead of loop folding can be used to increase the performance, or vice versa.
- For other basic blocks pipelined operation of vector statements can be used instead of normal parallel operation to save area, or vice versa.

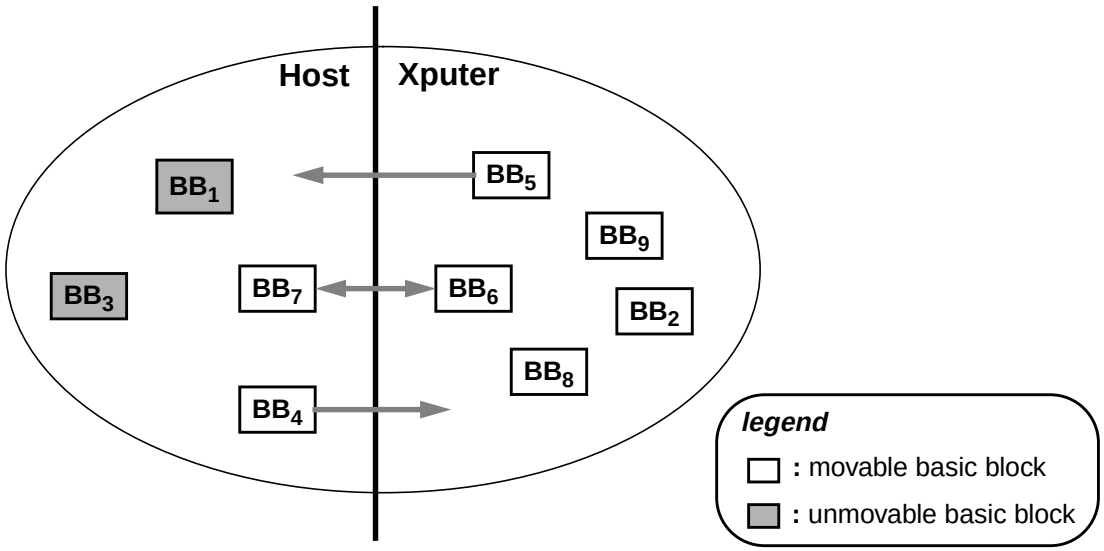

Figure 7. Partitioning into basic blocks that are executed on the host and basic blocks that are executed on the Xputer, possible exchanges are marked with grey shaded lines

The *COST*-function gives the complete cost of this partitioning by determining the overall execution time of the algorithm using the actual partitioning. The cost is determined from the profiler by adding the delays of the basic blocks evaluated on the host plus the delays of the basic blocks evaluated on the Xputer, plus the delay time for the synchronization, which includes a possible reconfigu-

ration time of the rALU of the Xputer. Due to the data dependencies the profiler takes concurrent evaluations into consideration. Moreover successive data dependent basic blocks may have a lower delay time when executed together on the same hardware platform than the addition of the delay times of the individual basic blocks. Such delay times are considered too. 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. 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 the second argument.

3.2 The Xputer Part

As shown in figure 3 the program part for the Xputer uses the X-C compiler and the datapath synthesis system (DPSS). The splitting into compiler and synthesis system allows simple adaptation of the framework if a different reconfigurable ALU is used.

3.2.1 The X-C compiler

The X-C compiler [Schm94] takes an almost complete subset of ANSI C as input. Only constructs, which would require a dynamic memory management to be run on the Xputer are excluded. These are pointers, operating system calls and recursive functions. 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. The X-C compiler is analyzing, restructuring and parallelizing the program part for the Xputer. It computes the parameter sets for the data sequencer and a rALU description for further synthesis in the datapath synthesis system, without user interaction. This realizes the second level of hardware/software co-design in the CoDe-X framework. First, the compiler performs a data and control flow analysis. The data structure obtained allows restructuring to perform parallelizations like those done by compilers for supercomputers (e.g. vectorization of statements). The next step performs a re-ordering of data accesses to obtain access sequences, which can be mapped well to the parameters of the data sequencer. Therefore, the compiler generates a so-called data map, which describes the way the input data has to be distributed in the data memory to obtain optimized hardware generated address sequences.

3.2.2 The Datapath Synthesis System

The datapath synthesis system (DPSS) allows to map statements from a high level language description onto the rDPA. The statements may contain arithmetic or logic expressions, conditions, and loops, that evaluate iterative computations on a small number of input data.

The task of configuring the rDPA is carried out in the following four phases: logic optimization and technology mapping, placement and routing, I/O 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 of DPUs with transparent chip boundaries.

Logic optimization and technology mapping. The condition statements are converted into single assignment code. The same is done for the loop conditions. The loop itself is controlled by the rALU controller. Loops and sequences of assignments are considered as basic blocks. Directed acyclic graphs (DAGs) are constructed from the basic blocks. Herewith common subexpressions, identical assignments, local variables, and dead code are removed. Further constant folding and reduction in strength is used. Unary operators are moved into the next operator if the operator library provides this new merged operator. This step reduces the number of required DPUs in the rDPA array. Further

parallelism of single expressions is increased by tree-height reduction. A simple algorithm is performed which use the commutativity and the associativity of some operators.

Placement and routing. A poor placement degrades the performance since some internal variables have to be routed via the internal I/O bus. During that time the bus is blocked for other I/O operations. A simulated annealing algorithm is chosen which gives better results than a simple constructive cluster growth algorithm, especially when the rDPA is connected as a torus. The simulated annealing algorithm belongs to probablistic algorithms which improve iteratively [Sher93]. The cost function considers the chip boundaries, the routing operators and with a high cost the connections via the internal I/O bus. Finally a kind of maze router is used for the final routing.

I/O scheduling. Due to the global I/O bus of the rDPA array, the loading of the data and the storing are restricted to one operation per time. An optimal sequence of these I/O operations has to be determined. Furthermore, if the statements of the algorithm belong to an inner loop and are executed several times in direct sequence, pipelining is used to improve speed. On the other side, if the statements belong to an outer loop and are executed once or several times but not in direct sequence, the vectorized expressions are used for different variables and thus improving area. This scheduling is done using a kind of list based scheduling algorithm known from high level synthesis [GDW92].

Code generation. The rDPA configuration file is computed from the mapping information of the processing elements and a library with the microprogram code of the operators. The configuration file for the rALU control unit is extracted from the final schedule of the I/O operators.

3.3 The Interface and the Host Part

Since the Xputer shares the memory with the host, the exchange of data is managed via the memory. Since the Xputer requires the data in a specific arrangement in the memory, described by the datamap description, the addresses of shared variables in the host program has to be updated accordingly. The host starts the Xputer by a special control signal which allows the Xputer to run on its own. The reconfigurations of the Xputer for every task can be performed in parallel to running host tasks before the starting signal. An interrupt generated by the Xputer signals the end of the computation. During that time, an algorithm on the host may also run concurrently.

At each time the Xputer is accessed, a synchronization point is integrated by the partitioner into the program to be executed on the host. The code at these points consists of operating-system calls for synchronizing parallel processes. Then the tasks for the host are compiled with the GNU C compiler.

4. Conclusions

CoDe-X, a two-level hardware/software co-design framework for Xputers has been presented. The Xputer is used as universal accelerator based on a reconfigurable datapath hardware. 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 the resource-driven sequential/structural partitioning. In the first level, performance critical parts of an algorithm are mapped onto the Xputer without manual interaction. The second level uses a resource-driven compilation/synthesis of the accelerator source code to optimize the utilization of its reconfigurable datapath resources.

The hardware/software co-design framework CoDe-X is completely specified. The X-C compiler and the datapath synthesis system (DPSS) have been implemented. The partitioner and the profiler are currently being implemented.

References

[AHR94] A. Ast, R. W. Hartenstein, H. Reinig, K. Schmidt, M. Weber: A General purpose Xputer Architecture derived from DSP and Image Processing; in M. A. Bayoumi (Ed.): VLSI Design Methodologies for Digital Signal Processing Architectures; Kluwer Academic Publishers, Boston, London, Dortrecht, pp. 365-394, 1994

[EHB93] R. Ernst, J. Henkel, T. Benner: Hardware-Software Cosynthesis for Microcontrollers; IEEE Design & Test, pp. 64-75, Dec. 1993

[FCCM94] Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, Napa, CA, April 1994

[GCM94] R. K. Gupta, C. N. Coelho, G. De Micheli: Program Implementation Schemes for Hardware-Software Systems: IEEE Computer, pp. 48-55, Jan. 1994

[GDW92] 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

[HHR91] R. W. Hartenstein, A. G. Hirschbiel, M. Riedmüller, K. Schmidt, M. Weber: A Novel ASIC Design Approach Based on a New Machine Paradigm; IEEE Journal of Solid-State Circuits, Vol. 26, No. 7, July 1991

[HKR94] R.W. Hartenstein, R. Kress, H. Reinig: A Dynamically reconfigurable Wavefront Array Architecture for Evaluation of Expressions; Int. Conference on Application Specific Array Processors, San Francisco, CA, pp. 404-414, August 1994.

[LWP94] 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

[PuKo89] P. Puschner, Ch. Koza: Calculating the Maximum Execution Time of Real-Time Programs; The Journal of Real-Time Systems p. 159-176, Kluwer Academic Publishers 1989

[Sher93] N. A. Sherwani: Algorithms for Physical Design Automation; Kluwer Academic Publishers, Boston 1993

[Schm94] K. Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph.D. Thesis, University of Kaiserslautern, 1994

[WoTs92] M. Wolfe, C.-W. Tseng: The Power Test for Data Dependence; IEEE Transactions on Parallel and Distributed Systems, Vol. 3, No. 5, Sept. 1992