A Two-Level Hardware/Software Co-Design Framework for Automatic Accelerator Generation

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

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 a novel hardware/software Co-Design framework CoDe-X for automatic generation of Xputer based accelerators. CoDe-X accepts C-programs and carries out both, the host/accelerator partitioning for performance optimization, and (second level) the parameter-driven sequential/structural partitioning of the accelerator source code to optimize the utilization of its reconfigurable datapath resources.*

Keywords: *automatic partitioning, profiling, Xputers, partitioning compilers, hardware/software trade-off, performance/area trade-off, accelerator generation*

1. Introduction

The movement of hardware/software co-design started several years ago as the changing nature of information processing systems pressured hardware and software engineers to work together more closely. The throughput of digital systems can be increased by a systematic partitioning of an application in hardware and software parts. Trade-off analysis - the shift of functionality from hardware to software and vice versa - was emphasized.

The scene of hardware/software co-design has introduced a number of approaches to speed-up performance, to minimize hardware/software trade-off and to reduce total design time [1], [4], [5]. Further co-design approaches are advocated by the FCCM-scene, 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 conveniently 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 (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 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 hardware/ software co-design framework CoDe-X. In this section the different strategies for the partitioning are shown and the used tools for evaluating the compilation and synthesis task are sketched. Section 4 discusses the partitioning of an computation-intensive example in the area of fractal images.

2. The Underlying Target Hardware

The target hardware consists of a host workstation that uses the Xputer as hardware accelerator. The key concept of an Xputer [2], 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.

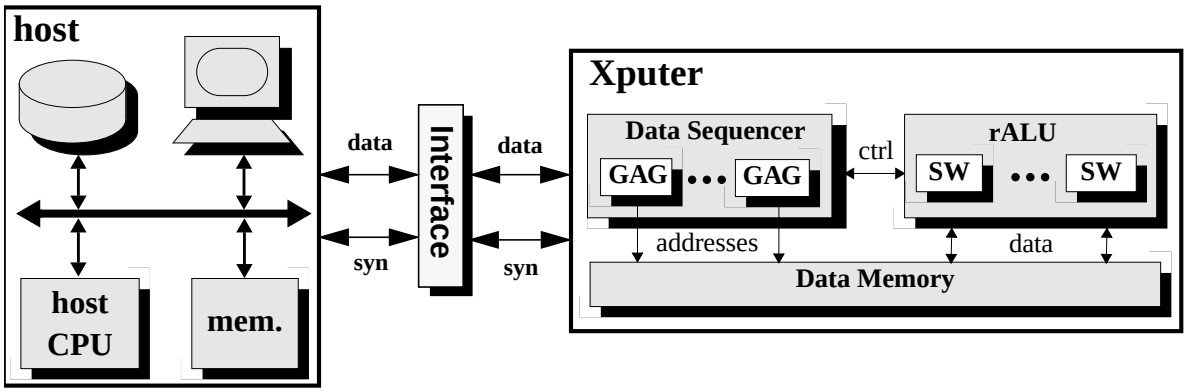

Figure 1. Block diagram of the underlying hardware

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 [3].

3. 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 [2]. 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

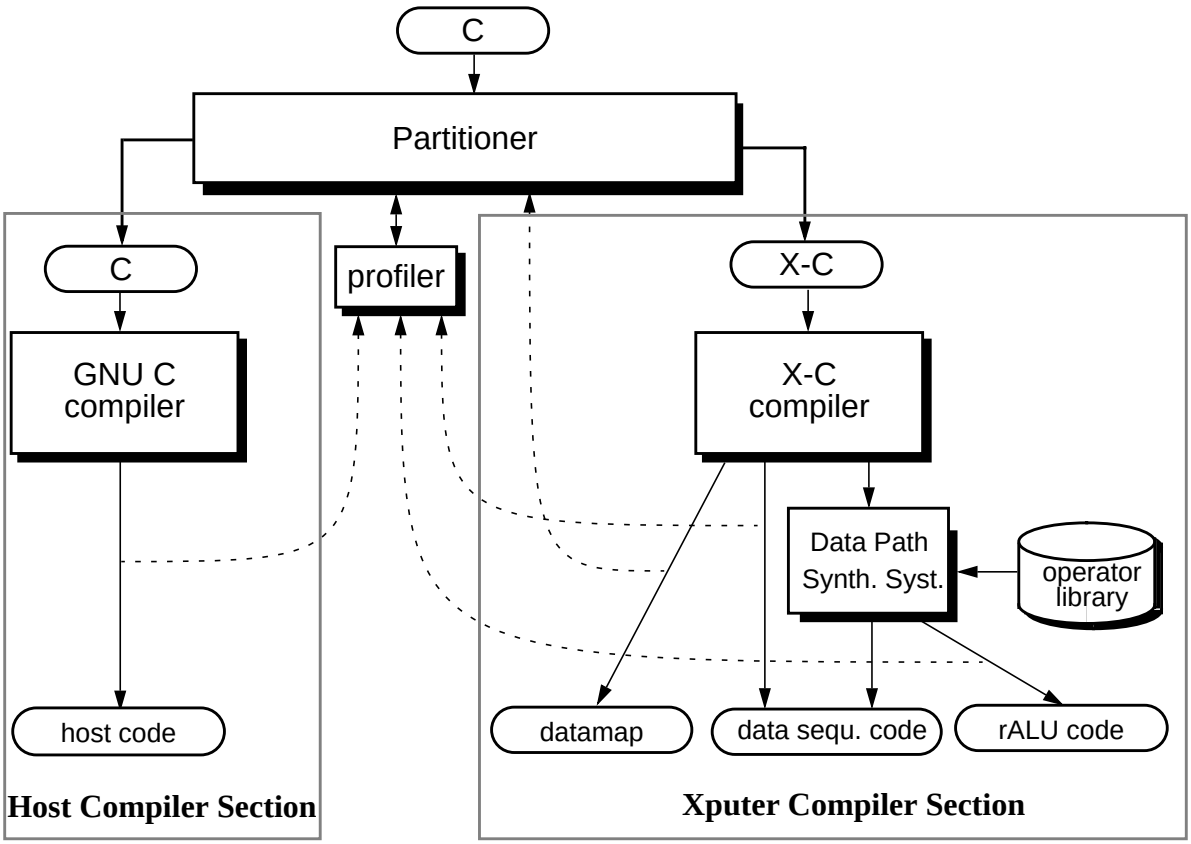

Figure 2. Overview on the CoDe-X environment

the second level in a 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) [3]. 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 data distribution (datamap). The description of the datamap gives the location of the variables in the memory. The partitioner is including the addresses of shared variables and shared arrays into the C code for the host in order to compute on the same memory. Thus the datamap 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 2 gives a detailed overview on the hardware/software co-design framework CoDe-X.

3.1 The Profiling-driven Iterative Partitioning

The partitioner in the first level represents the input C program as a graph, called flow graph. This flow graph is first analyzed for code-segments without dynamic structures and operating system calls e. g. I/O-routines, because these segments are well suited for the

Xputer. This is performed by a preprocessor within the partitioner of CoDe-X (figure 2). Then these segments are partitioned into basic blocks (BBs, grey shaded in figure 3), which can later be executed on the host or on the Xputer. 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. The remaining code segments are host-specific (for example I/O routines) and they are also partitioned into basic blocks (H-BBs in figure 3), which will later be executed by the host in every case, because they are not efficient for the Xputer. Figure 3a shows an example of a graph representation of basic blocks with their control flow. A data dependency analysis based on the GCD-test [9] gives the data dependencies between the basic blocks (figure 3b). 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 compound BB, 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. Another possibility for achieving a higher performance is to unroll the innermost loop, until the Xputer hardware constraints are reached. 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 3c). The basic blocks, that can be evaluated concurrently, are displayed side by side.

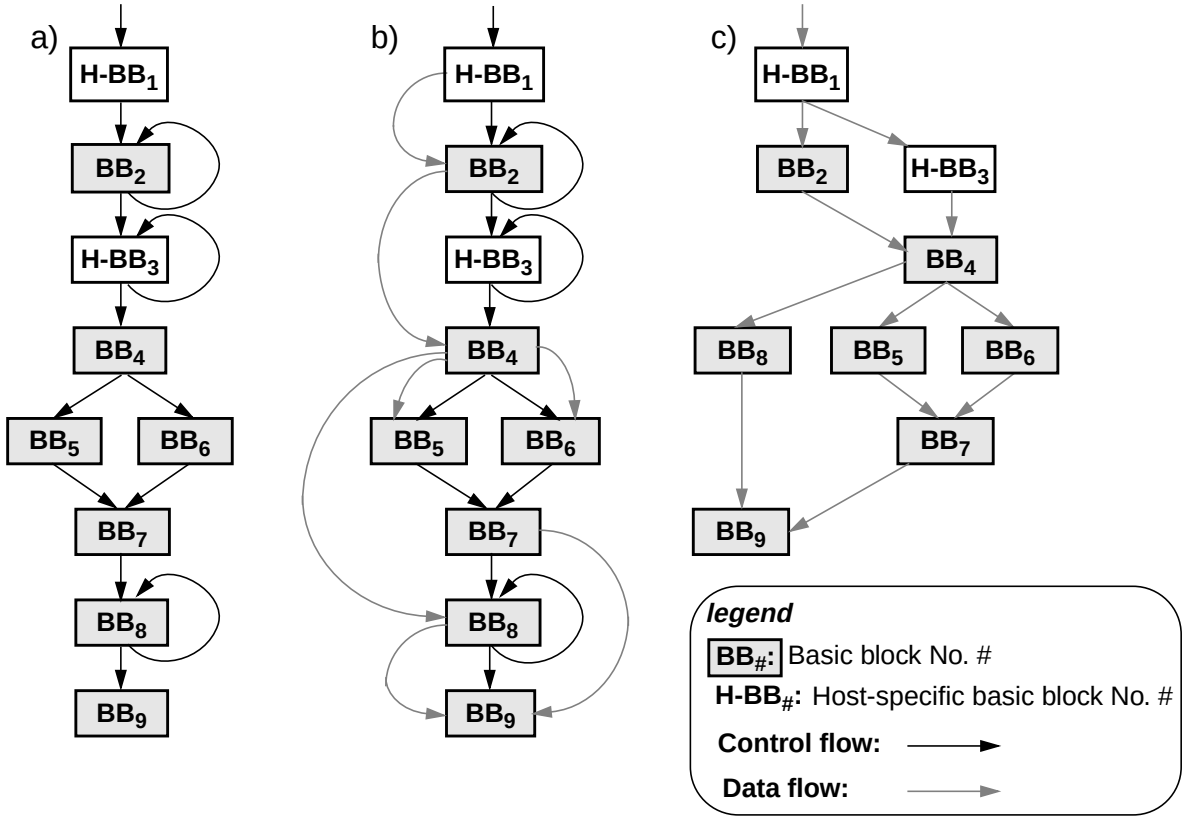

Figure 3. Graph representation of the basic blocks: a) control flow only, b) with data dependencies, c) data flow only showing possible parallelism between basic blocks

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 and also for the compound BBs the time for evaluation on the Xputer as well as the evaluation on the host is approximately computed (cost function of an BB). 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 4 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.

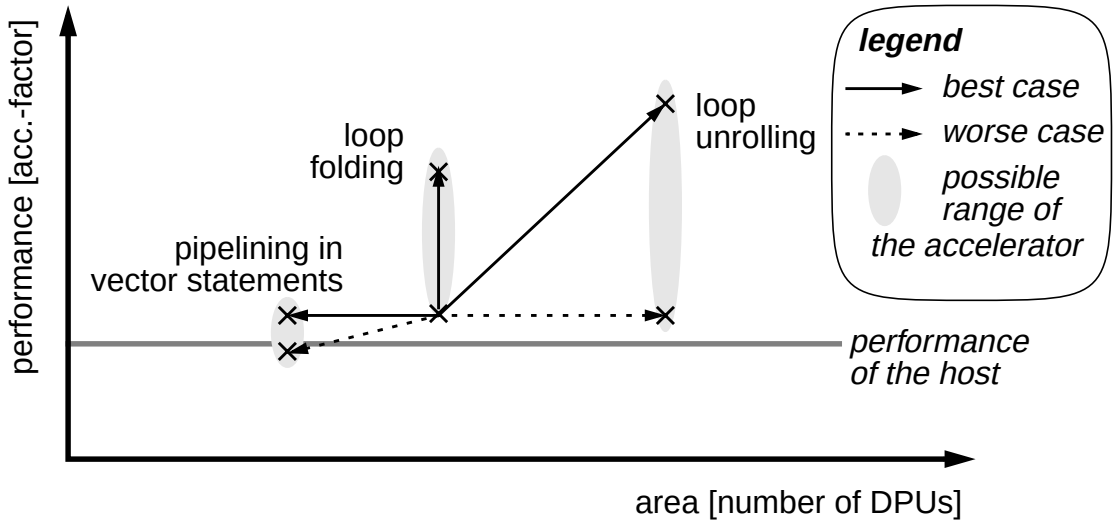

Figure 4. Example of a performance/area trade-off of the basic blocks

The performance of the host is approximated by examining the code using branch prediction techniques [8]. 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 [11]. Figure 4 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:

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

Partitioner. The partitioning phase in the first level is based on a simulated annealing algorithm [7], which is controlled by the profiler-computed cost functions of the BBs. 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.

3.2 The Resource-driven Partitioning of the X-C Compiler

As shown in figure 2 the program part for the Xputer uses in the second level of partitioning the X-C compiler and the datapath synthesis system (DPSS). The splitting into compiler and synthesis system allows simple adaptation for different reconfigurable ALUs.

The actual X-C compiler [6] 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 (software part) and a rALU description for further synthesis in the datapath synthesis system (hardware part), without user interaction. This realizes in the second level of hardware/software co-design a constructive and resource-driven partitioning. 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.

The datapath synthesis system (DPSS) [3] 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.

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 code-segments to be executed on the host. The code at these points consists of

operating-system calls for synchronizing parallel processes. The tasks for the host are compiled with the GNU C compiler.

4. Example: The Mandelbrot Set of Fractal Images

The subject of chaos and fractals [12], [10] has come into popular prominence in the last years due to the fact, that the computation-intensive fractals has implications for us all, in areas as next week weather, Jupiter´s famous red spot, heart attacks and liver diseases.

The function that is iterated to give the Mandelbrot set is very simple (equation (2)):

$$z_{n+1} = z_n^2 + c \tag{2}$$

where z is a complex number, which is initialised to a starting value and then is allowed to alter as the function is iterated, and c is a complex number, that stays constant for the sequence of iterations. If we want to compute the Mandelbrot set (black/white), the computer screen is first splitted into a grid (size of fractal image), where real numbers are represented on the x-axis and imaginary numbers on the y-axis. Then a point on this grid is selected representing a particular complex number, which we called c. The real part is being represented by the x position and the imaginary part by the y position. Now another complex number, called z, is set to 0 and equation (2) is iterated until the absolute value of z goes off to infinity (behaving chaotically) or, after a pre-specified number of iterations, is still fairly small (tending towards a specific attractor). If the function has iterated to infinity after a number of steps, then plot the point on the screen corresponding to the real and imaginary values of c in “white”. If the number is still small after iterating it for a pre-determined number of steps, then we assume that this value of c will not iterate to infinity and the corresponding point is coloured “black”. This process generates the Mandelbrot set.

Our small example was divided in four basic blocks. Two of them were filtered in the first step of the iterative partitioner for being executed on the host in every case (H-BB_1, H-BB_4: I/O-routines). The remaining two basic blocks (BB_2, BB_3) are two nested for loops (within a while-loop inside the inner for-loop for computing the complex Mandelbrot set) are potential candidates for mapping on the Xputer, because the preprocessor could transform in equivalent X-C code. The profiler is computing the host- and Xputer-cost functions for them. The data dependencies require a sequential execution of the four basic blocks, which will influence the overall execution time of this application. In the following simulated annealing process different partitions have been analyzed. The final decision was to merge BB_2 and BB_3 to a single compound basic block and to shift this block to the Xputer. The partitioning overhead for synchronization and data transfers was to inefficient for computing for example BB_2 on the host and executing BB_3 on the Xputer. The X-C compiler on the second level of hardware/software co-design is then mapping the compound basic block on the Xputer. In this final solution the Xputer must be configured only one time. The address generation of the outer for-loop is handled by one generic address generator (GAG, see figure 1) and the inner for-loop by another GAG. During execution time a controler within the data sequencer will switch from one GAG to the second GAG without reconfiguration. The while-loop within the inner for-loop is mapped directly onto the rALU. In different simulation examples with this application we achieved acceleration factors of up to 6 in comparison with a SUN SPARC 10/51.

5. 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 localized and 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. In several application examples good acceleration factors have been achieved in comparison to a single workstation without accelerator-board.

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

[1] K. Buchenrieder, C. Veith: CODES: A Practical Concurrent Design Environment. Workshop handouts 1st Int'l Workshop on Hardware/Software Co-Design, Estes Park, Colorado, 1992.

[2] Reiner W. Hartenstein, Jürgen Becker, Rainer Kress, Helmut Reinig, Karin Schmidt: A Reconfigurable Machine for Applications in Image and Video Compression. Conference on Compression Technologies and Standards for Image and Video Compression, Amsterdam, The Netherlands, March 1995.

[3] 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.

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

[5] W. Luk, T. Lu, I. Page: Hardware-Software codesign of Multidimensional Programs. Proc. of the IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, Napa, CA, April 1994.

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

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

[8] Thomas Ball, James R. Larus: Branch Prediction for free. Proc. of the ACM-SIGPLAN ´93: Conference on Programming Language Design and Implementation, pp. 300 - 313; ACM-Press, 1993.

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

[10] Joe Pritchard: The Chaos Cookbook. Butterworth-Heinemann, UK, 1992.

[11] 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.

[12] Heinz-Otto Peitgen, Dietmar Saupe: The Science of Fractal Images. Springer-Press, New-York, 1988.