

CoDe-X: A Novel Two-Level 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 (second level) the resource-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 minimize hardware/software trade-off and to reduce total design time [2], [4], [8]. Further co-design approaches are advocated by the FCCM-scene [3], 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 efficiently support “soft” hardware: because of its extremely tight coupling between instruction sequencer and ALU. So a new paradigm is desirable 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 uni-

versal 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 hardware/software co-design framework CoDe-X and its strategies. Section 4 discusses the partitioning of an computation-intensive example in the area of fractal images.

2. The Underlying 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 [6], 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 this sequence by a few 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



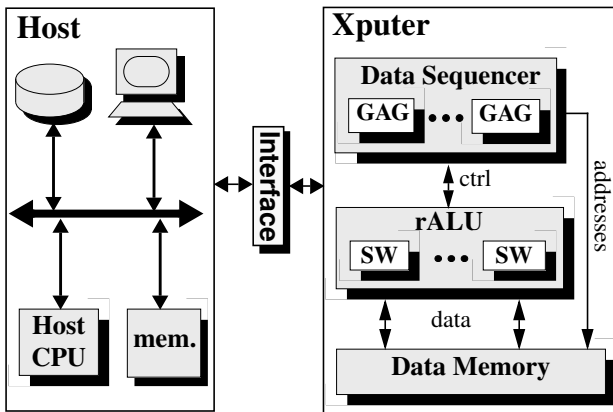


Figure 1. Block diagram of hardware platform

the scan windows. This address computation does not require memory cycles. That's why it is an accelerating factor.

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 [7].

3. The CoDe-X Framework

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 and a part for execution on the accelerator. In this case the Xputer is used as accelerator [6]. 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 the second level in a sequential part for programming the data sequencer, and a structural part for configuring the rALU. This rALU

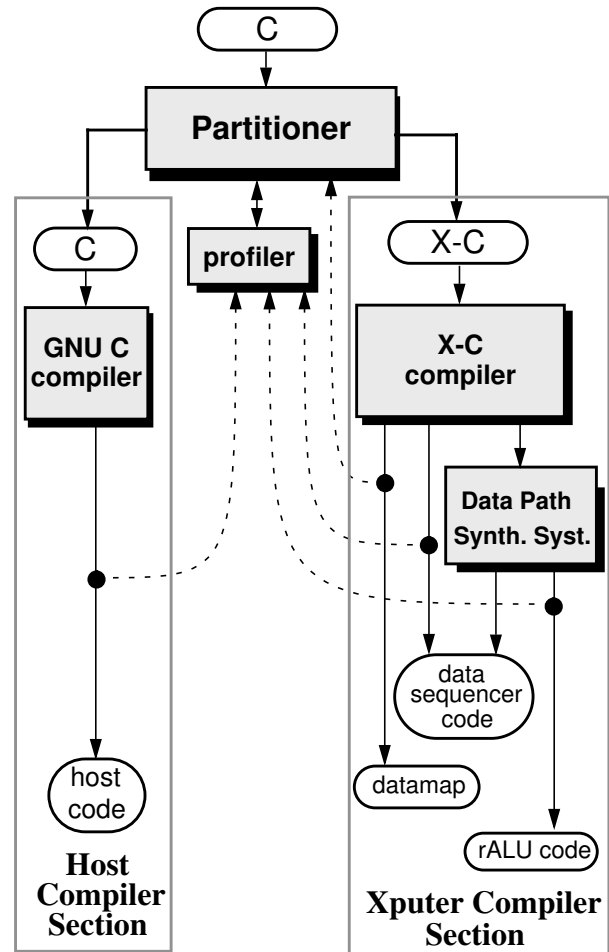


Figure 2. Overview on the CoDe-X environment

description is used for further synthesis in the datapath synthesis system (DPSS) [7]. The output of the X-C compiler results in three files: a rALU file for the configuration of the rALU (rALU code, see Figure 2), a parameter set for the loading of the data sequencer (data sequencer code), and a description of the datamap. 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.

3.1 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 an X-C filter within the partitioner of CoDe-X (Figure 2). Then these segments are partitioned into basic blocks (BBs), which can later be executed on the host or on the Xputer. A data dependency analysis based on the GCD-



test [13] gives the data dependencies between the basic blocks. 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 perform different code transformations like unrolling the innermost loop, until the Xputer hardware constraints are reached, or like transforming a large loop into several smaller loops, which can be mapped as parallel basic blocks onto the Xputer.

The partitioning into parts for the host and parts for the Xputer is based on the performance analysis of a profiler (Figure 2). 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 every 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 performance of the host is approximated by examining the code using branch prediction techniques [1]. 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].

The iterative partitioning phase in the first level of CoDe-X is based on a simulated annealing algorithm [5], 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 randomly. If the new partitioning results in a lower overall cost (due to the data dependencies the profiler takes concurrent evaluations into consideration), 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 Resource-driven partitioning by X-C compiler

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

The actual X-C compiler [12] 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. 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) [7] 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 goes via memory. Since the Xputer data requires a specific storage scheme, described by the datamap, the addresses of shared variables in the host program has to be updated accordingly. The host starts the Xputer by a 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 iterative 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: Mandelbrot set of fractal images

The subject of chaos and fractals has come into popular prominence in the last few years e. g. the Mandelbrot set [10], which has implications for us all, in areas as next week weather, Jupiter's famous red spot and heart attacks and was actually born thirty years ago [9].

The function that is iterated to give the Mandelbrot set is very simple (equation (1)):

$$z_{n+1} = z_n^2 + c \quad (\text{eq. 1})$$

where z is a complex number, which is initialized 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.

The C-program of our small example was divided into 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 (I/O-routines). The remaining two basic blocks are two nested for-loops (within a while-loop inside the inner for-loop for computing the complex Mandelbrot set). They are potential candidates for mapping onto the Xputer, because the preprocessor could transform them in equivalent X-C code. The profiler has computed 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 the two nested for loops 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 one loop on the host and executing the other on the Xputer. The X-C compiler in the second level of hardware/software co-design was then mapping the compound basic block onto the Xputer. In this final solution the Xputer has to 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 controller 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 simulation examples with this application we achieved acceleration factors of up to 6 in comparison to a SUN SPARC 10/51.

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 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 shifted to 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 an accelerator-board.

6. References

- [1] T. Ball, J. R. Larus: Branch Prediction for free; Proc. of the ACM-SIGPLAN '93: Conf. on Programming Lang. Design & Implement., pp. 300 - 313; ACM-Press, 1993
- [2] R. Ernst, J. Henkel, T. Benner: Hardware-Software Cosynthesis for Microcontrollers; IEEE Design & Test, pp. 64-75, Dec. 1993
- [3] Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, Napa, CA, April 1994
- [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] L. Lin: High-Level Synthesis, Introd. to Chip and Syst. Design; Kluwer Acad. Pub., Boston, London, 1992
- [6] R. W. Hartenstein, J. Becker, R. Kress, H. Reing, K. Schmidt: A Reconfigurable Machine for Applications in Image and Video Compression; Conf. on Compression Techn. & Standards for Image & Video Compression, Amsterdam, Netherlands, March 1995
- [7] 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
- [8] 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
- [9] Heinz-Otto Peitgen, Dietmar Saupe: The Science of Fractal Images; Springer-Press, New-York, 1988
- [10] Joe Pritchard: The Chaos Cookbook; Butterworth-Heinemann, UK, 1992
- [11] 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
- [12] K. Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph.D. Thesis, University of Kaiserslautern, 1994
- [13] M. Wolfe, C.-W. Tseng: The Power Test for Data Dependence; IEEE Transactions on Parallel and Distributed Systems, Vol. 3, No. 5, Sept. 1992

