Reiner W. Hartenstein, Jürgen Becker, et al.: A Profiling-driven Hardware/Software Partitioning of High-level Language Specifications; IFIP Int. Workshop on Logic and Architecture Synthesis, Grenoble, France, December 1995

A Profiling-driven Hardware/Software Partitioning of High-level Language Specifications

Reiner W. Hartenstein, Jürgen Becker, Rainer Kress
University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de

Abstract

The profiling-driven hardware/software partitioning of high-level language specifications is presented in this paper. It is performed by a hardware/software co-design framework (CoDe-X), 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. Introduction

Earlier hardware/software co-design approaches have introduced a number of systems to speed-up performance, to minimize hardware/software trade-off and to reduce total design time [EHB93], [GCM94], [LWP94] and others. Further co-design research advocates the so-called Custom Computing Machines (CCMs) approach [FCCM95], 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. It is already difficult to add one or two extra instructions to the given instruction set. The von Neumann paradigm does not support efficiently “soft” hardware due to its extremely tight coupling between instruction sequencer and ALU. So a new paradigm is required like the one of Xputers [HaKr95], which conveniently supports “soft” ALUs like the rALU concept (reconfigurable ALU). Furthermore most addressing and control overhead is eliminated by a data sequencer which gives hardware support for a rich repertory of data access sequences. An Xputer is a reconfigurable, parallel and data-driven architecture. For this new class of hardware platforms a new class of compilers is needed, which generate both, sequential and structural code: restructuring 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: profiling-driven host/accelerator partitioning for optimizing performance and a structural/sequential partitioning for optimizing the hardware/software trade-off of the Xputer resources. This paper introduces the profiling-driven partitioning performed in the first level of 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.

The paper is organized as follows: First the underlying hardware is explained. Section 3 presents the CoDe-X framework focusing on the profiling-driven partitioning of C-specifications. The different algorithms and strategies for the partitioning are shown. Section 4 discusses the partitioning of an computation-intensive example in the area of image processing.

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], [HaKr95] is to map efficiently performance critical algorithms onto the reconfigurable hardware architecture. Performance critical algorithms typically apply the same set of operations to a large amount of data. So the main applications are numeric and scientific computations, which have regular data dependencies in their algorithms for exploiting the Xputer hardware features efficiently. 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 a two-dimensional organized 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 up to seven hardwired generic address generators (GAGs), which are capable of interpreting such parameter sets to compute generic address sequences for the scan windows. These seven GAGs can operate in parallel [AHR94]. 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. 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.

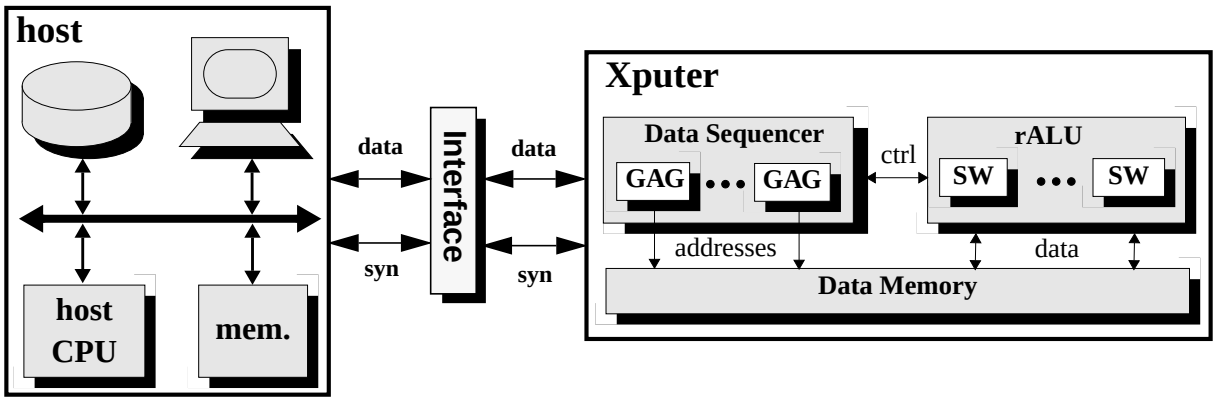

Figure 1. Block Diagram of the underlying hardware

Our most recent prototype of the Xputer operates as a 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 [HaKr95].

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 program is partitioned in a first level into a part for execution on the host and a part for execution on the Xputer, which is used as accelerator in this case. 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 (see Figure 2). This rALU file 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 (data sequencer code), and a description of the storage scheme (datamap). The description of the storage scheme 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 2 gives a detailed overview on the hardware/software co-design framework CoDe-X.

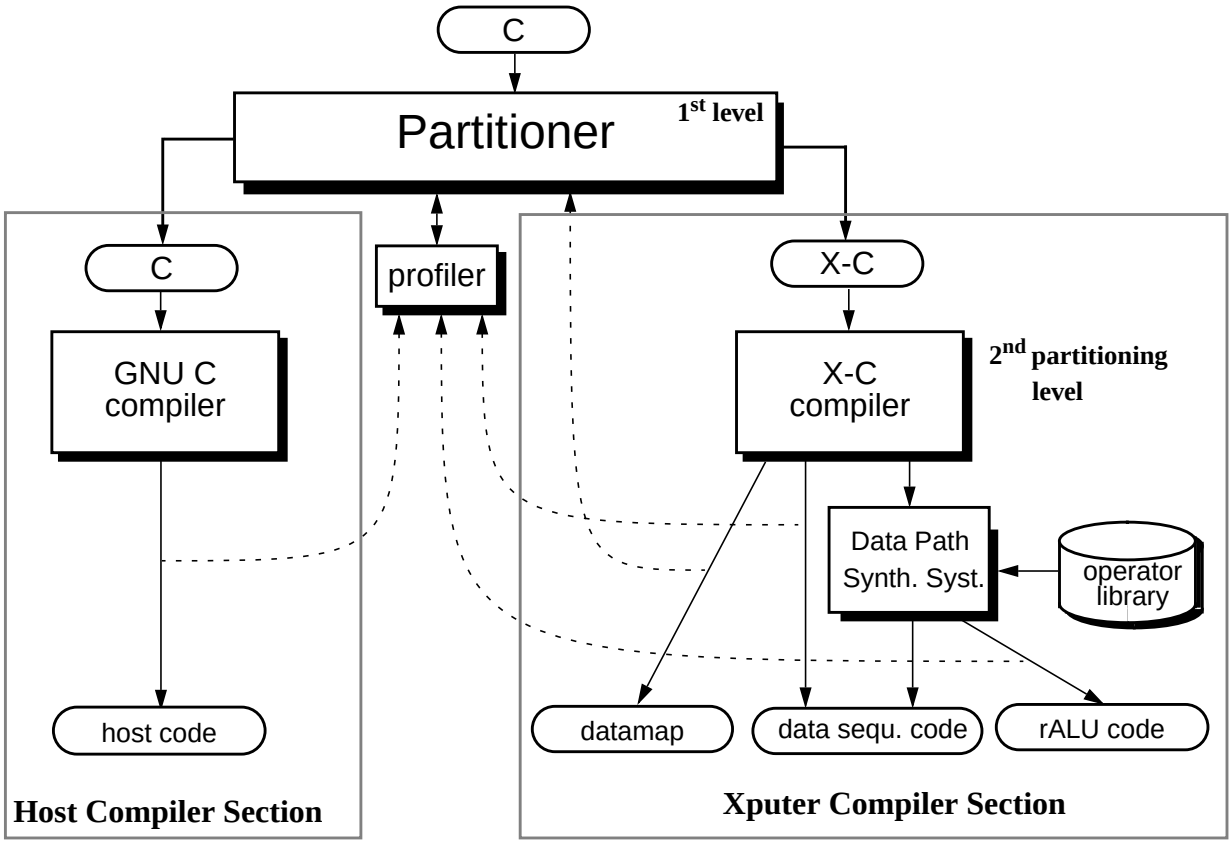

Figure 2. Overview on the CoDe-X environment

3.1 Partitioning of C-Specifications based on Profiling

The partitioner in the first level represents the input C program as a graph, called flow graph. This flow graph is first filtered for code-segments without dynamic structures, operating system calls and I/O-routines, because these filtered segments are candidates for the Xputer. This is performed by a preprocessor within the partitioner of CoDe-X (see Figure 2). Then these segments are partitioned into Xputer-specific basic blocks, which can later be executed on the host or on the Xputer. An

Xputer-specific basic block is a sequence of consecutive statements, which can be executed by a so called Xputer-task (represented by one rALU configuration, an application-dependent data map and one ore more corresponding scan pattern). The remaining code segments are host-specific (for example I/O routines), which will later be executed by the host in every case. A data dependency analysis based on the GCD-test [WoTs92] gives the data dependencies between the basic blocks.

Code Optimizations. In this phase some optimizations can be introduced that reduce the number of Xputer-tasks (XTs). Basic blocks in nested loops, which are computed in different Xputer-tasks, may be merged together to one single Xputer-task. This heuristic is based on the knowledge that nested loops achieve high performance factors on Xputers. The area required by the reconfigurable datapath architecture can be approximated by the number of operators in the statement blocks of these loops.

Another optimization can be the transformation of one large single nested loop into a double nested loop by organizing the computation in the original loop into chunks of approximately equal size. This code transformation is called strip mining ([Love77], Figure 3) and is used here for com-

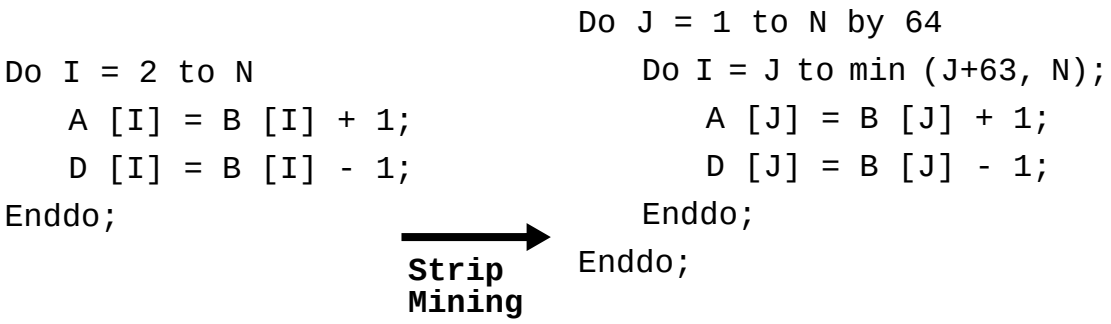

Figure 3. Example for strip mining with a block size of 64

puting these chunks probably in parallel on the Xputer. This technique can be well used in image processing applications, if you divide an image into stripes of equal sizes in order to manipulate these stripes concurrently.

The transforming of two adjacent loops into a single one (loop fusion, [Love77]) can be also performed in this phase for reducing the amount of loop overhead in a program. This technique is useful and performed in our approach, if the two loops cannot be computed in parallel and if the resulting single loop can be computed by one Xputer-task.

Moreover loop unrolling, up to the degree that the unrolled statement body fits onto the reconfigurable hardware resources, can be performed. The DPSS (see Figure 2) is applying loop folding (pipelining across loop boundaries), if this is possible, because it requires no more hardware resources.

The optimizations provided by CoDe-X allow to realize several implementations of an Xputer-task for the reconfigurable datapath architecture with different performance/area trade-offs, which is called design space. Figure 4 marks the regions that can be reached with the different optimization techniques. For each Xputer-task several acceleration factors can be computed. The initial point in the design space requires *n* operators or rather DPUs for implementing the function of this task. If loop folding is possible, the performance (acceleration factor) increases up to *max. fold* without requiring more DPUs. If pipelining in vectorized statements can be performed, the performance is the same (*max. pipe*) in the best case or the acceleration factor is decreasing. But the number of used DPUs will be less. The highest increase of acceleration factors are gained by loop unrolling and strip mining (*max. unroll*, *max. strip*), which require the most hardware resources of the proposed optimization techniques. If e. g. two iterations of a loop body will be unrolled, then *2n* DPUs are needed (see Figure 4). The values of *max. fold*, *max. pipe*, *max. unroll* and *max. strip* depend on the actual viewed Xputer-task.

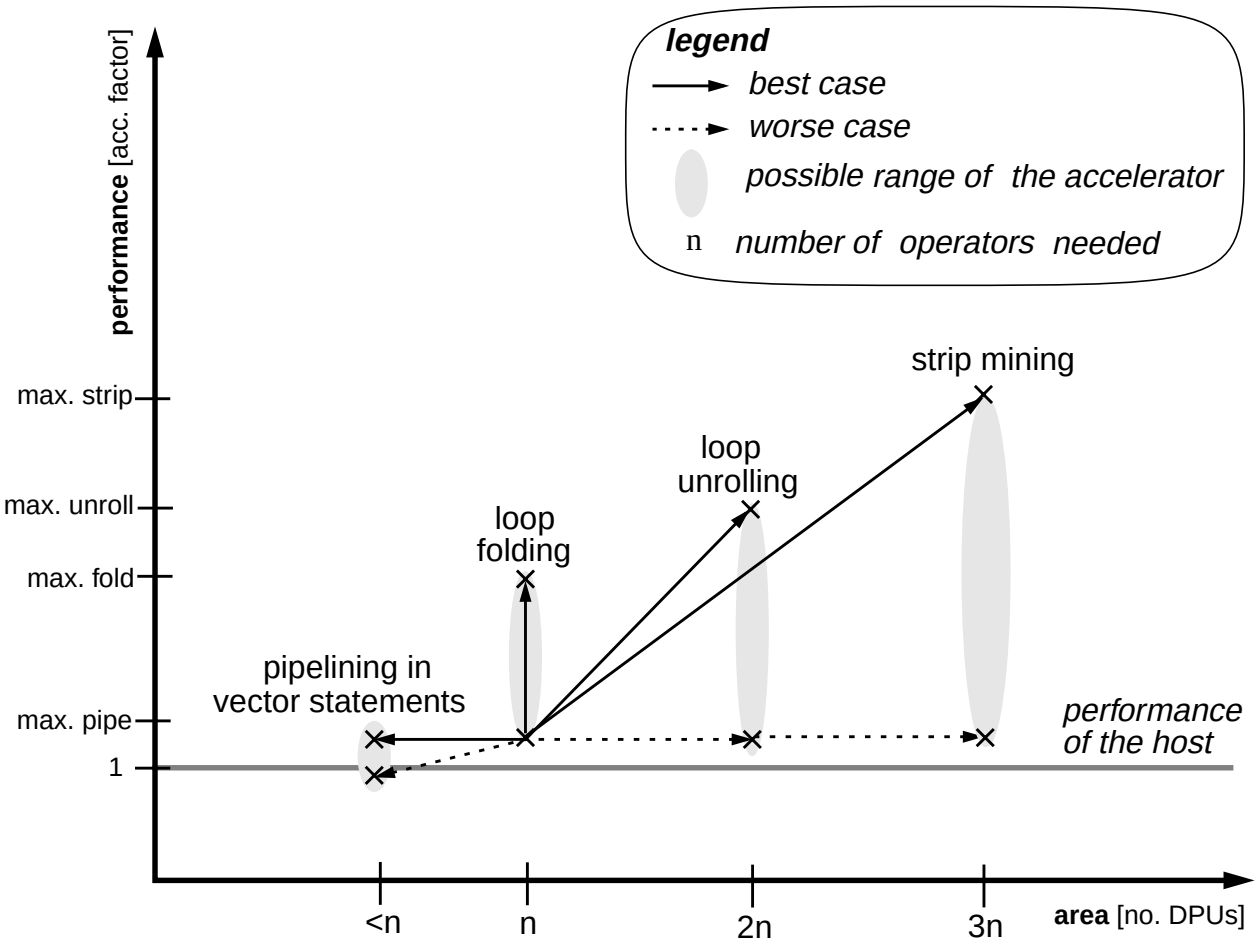

Figure 4. Example of performance/area trade-off for an Xputer-task

For each point in the design space that can be reached with the Xputer implementation the acceleration factor is computed as:

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

Of course the proposed methods can be combined to reach further positions in the design space.

Rearranging the tasks according to their data dependencies results in a data flow graph, where the tasks are represented by the nodes. Independent nodes may be executed in parallel, dependent on the final scheduling of the tasks.

Profiler. The partitioning into parts for the host and parts for the Xputer is based on the performance analysis of a profiler. For each Xputer-task (candidates for possible execution on the Xputer) the time for evaluation on the Xputer as well as the evaluation time on the host is approximately computed. The performance data for the Xputer task is received from the datapath synthesis system (DPSS) [HaKr95] and the X-C compiler [Schm94]. The X-C compiler is able to evaluate the number of iterations for the statement blocks. Knowing the worst case execution time for the statement block from the datapath synthesis system, the complete evaluation time can be approximated. The time-consuming placement of operators in the DPSS does not have to be performed for approximating the execution time of the statement blocks. The profiler is computing different execution times for every Xputer-task dependent on different points in the design space.

The performance of the host is approximated by examining the 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]. Branch prediction techniques [BaLa93] are considered by computing the execution time of a code segment. Figure 4 shows a grey shaded line giving the performance of the host.

The profiler is also computing the overall execution time of an algorithm, which is used as *COST*-function in the iterative partitioning process (see simulating annealing below). This function is called $COST_{part}$ and gives the complete cost by determining the overall execution time of the algorithm using the actual partitioning. The cost of a partitioning is determined from the profiler according to equation (2):

$$COST_{part} = \sum_{i \in HT} t_{exe_i} + \sum_{j \in XT} t_{exe_j} + \sum_{j \in XT} t_{reconf} + \sum_{\forall XputerCalls} t_{syn} - \sum_{\forall XT} t_{overlap} \qquad (2)$$

whereas:

$\sum_{i \in HT} t_{exe_i}$: sum of execution times of tasks (HTs), which will be executed on the host

$\sum_{j \in XT} t_{exe_j}$: sum of execution times of Xputer-tasks (XTs), which will be executed on Xputer

$\sum_{j \in XT} t_{reconf}$: sum of delay times for reconfiguring the Xputer during run time

$\sum_{\forall XputerCalls} t_{syn}$: sum of delay times for synchronizing host/Xputer (operating system calls)

$\sum_{\forall XT} t_{overlap}$: sum of overlapping execution times between Xputer-tasks executed simultanously on host/Xputer

Due to the data dependencies the profiler takes concurrent evaluations into consideration. The profiler also considers possible delay times for reconfiguring the Xputer during run time. This will be necessary, if a partition moves more Xputer-tasks to the Xputer, than parallel GAG- or rather rALU-units are available. If possible, the reconfigration is performed in parallel to running host tasks.

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 Xputer-tasks 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.

For faster access all profiler computed delay times of the Xputer-tasks 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 Xputer-tasks are not considered. 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.

3.2 Resource-driven Partitioning of the Xputer Part

As shown in figure 2 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.

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. 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 file for further synthesis in the datapath synthesis system (see Figure 2), without user interaction. This realizes the resource-driven partitioning of the second level of hardware/software co-design in the CoDe-X framework. Therefore, the compiler generates also 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) 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. Further details about the four phases of the DPSS, as well as examples of the mapping onto the rDPA can be found in [HaKr95].

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 have to be updated accordingly. The host starts the Xputer by a special control signal which allows the Xputer to run on its own. The execution of Xputer-tasks and necessary reconfigurations of the Xputer hardware can be performed in parallel to running host tasks. An interrupt generated by the Xputer signals the end of the computation.

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

4. Example: Smoothing Algorithm for Edges

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 [GoWi77]. 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:

$$g(x, y) = \frac{1}{M} \sum_{(n, m) \in S} f(n, m) \qquad (3)$$

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. The structure of the correspond-

ing C-program is illustrated in Figure 5. This small example for illustrating the methods of CoDe-X is divided in four tasks. Two of them were filtered out in the first step of the iterative partitioner for being executed on the host in every case (HT_1, HT_2: I/O-routines). The remaining two tasks (XT_1,

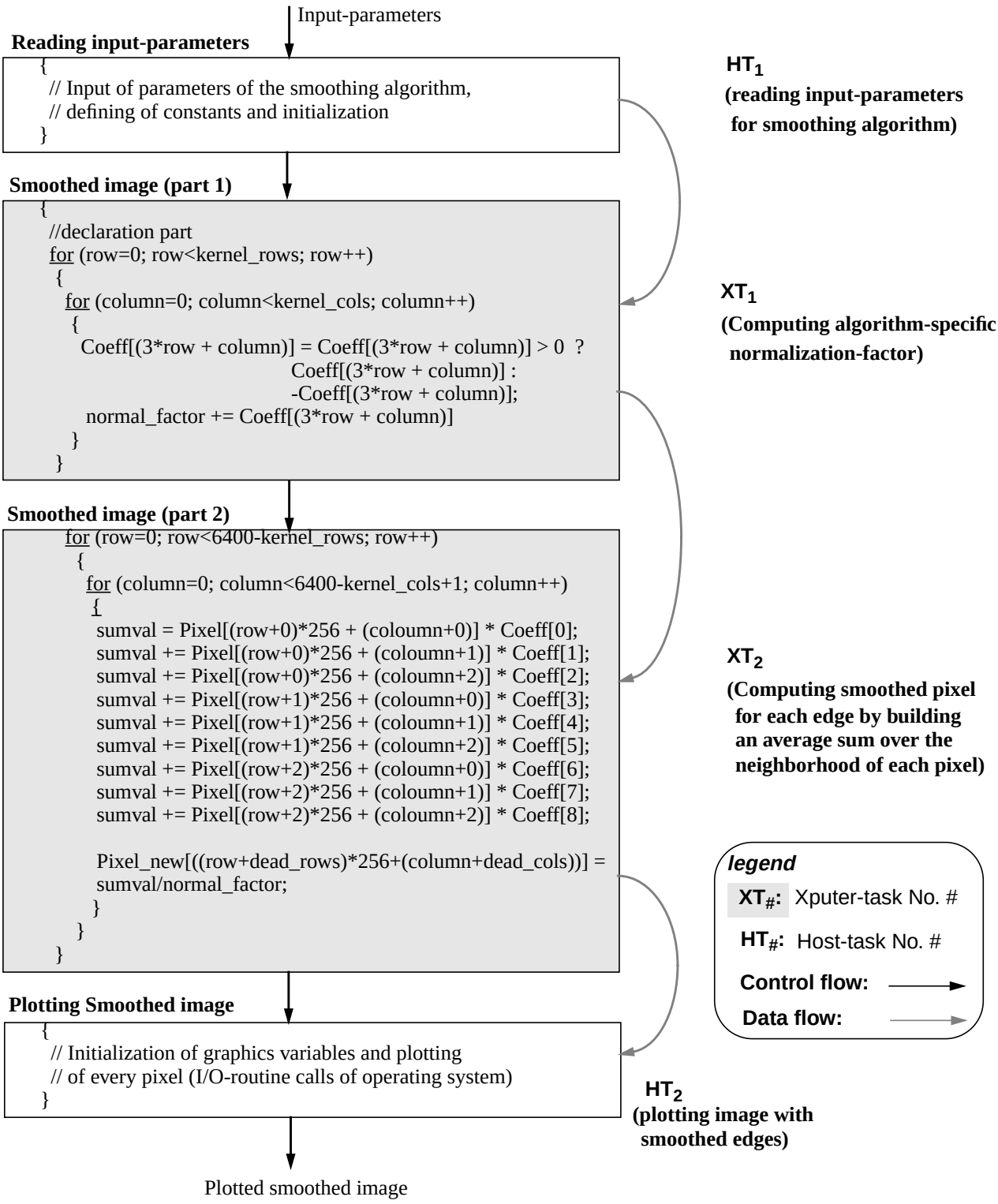

Figure 5. Tasks of the smoothing algorithm example incl. their control flow and data dependencies

XT_2) are potential candidates for mapping onto the Xputer. The profiler is computing the host- and Xputer- cost functions for them. The data dependencies require a sequential execution of the four tasks, which will influence the overall execution time of this application. The final decision was to shift both XT_1 and XT_2 to the Xputer. The X-C compiler on the second level of hardware/software co-design is then mapping these Xputer-tasks onto the Xputer. In this final solution the Xputer must be configured only once. The address generation for the data accesses in the fully nested loop of XT_1 is handled by one generic address generator (GAG, see Figure 1) and XT_2 by another GAG. During run time a controller within the data sequencer will switch from one GAG to another without reconfiguration. The performance/area trade-off of XT_2 is illustrated in Figure 6 and was derived by implementing this block on a SUN SPARC 10/51 (287,0 seconds) and by analyzing its implementation on the current Xputer prototype (unrolled version in 19,6 seconds, assuming 120 ns memory access time). This results in an acceleration factor of 14,6. By additionally dividing the image into stripes applying the strip mining code optimization technique (see section 3.1) for XT_2, the parallel Xputer hardware resources can be exploitet in an optimized way. Here this large fully nested loop over the image will be transformed into a number of smaller independent loops, whose will be manupulated by different independent GAGs and rALUs in parallel. So the first acceleration factor can be multiplied by the factor 5. This results in the final acceleration factor of 73,0 (see Figure 6). All necessary code optimiztations and transformations will be performed by the iterative partitioner in the first level of CoDe-X in this case.

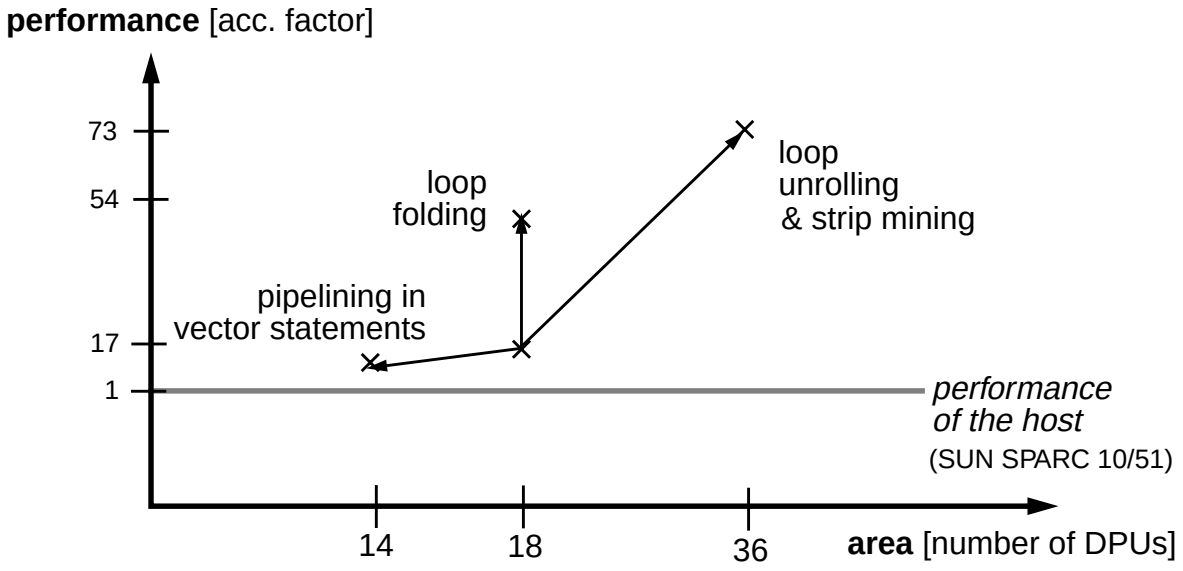

Figure 6. Performance/area trade-off of XT_2 inside of the smoothing algorithm (Figure 5)

In this example the communication overhead between host and Xputer is very small, because the image data can be accessed by both hardware platforms in the common memory and has not to be transfered. Due to this fact applications from image processing are well suited for our accelerator.

5. Conclusions

A profiling-driven hardware/software partitioning of high-level language specifications has been presented. CoDe-X, a two-level hardware/software co-design framework for Xputers is performing this in its first level. Here performance critical parts of an algorithm are identified and mapped onto the Xputer without manual interaction. 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. The second level uses a resource-driven compilation/synthesis of the accelerator source code to optimize the utilization of its reconfigurable datapath resources. The CoDe-X framework works fully automatic with no further user interaction. The current prototype of the accelerator is best suited for image and signal processing algorithms due to the two dimensional organization of the data memory, but not limited to such applications.

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

[BaLa93] 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.

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

[FCCM95] Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, FCCM'94, Napa, CA, April 1995

[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

[GoWi77] R.C. Gonzalez, P. Wintz: Digital Image Processing; Addison-Wesley Publishing Company, USA 1977

[HaKr95] 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

[HBK95] 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

[HHW90] 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 memorating the 30th Anniversary of the Computer Society of Japan, Tokio, Japan, 1990

[Love77] 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

[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