Performance-Evaluation in Xputer-based Accelerators

R. W. Hartenstein, J. Becker, K. Schmidt
University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de

Abstract

The paper presents the performance analysis process within the parallelizing compilation environment CoDe-X for simultaneous programming of Xputer-based accelerators and their host. The paper introduces briefly its hardware/software co-design strategies at two levels of partitioning. CoDe-X performs both, at first level a profiling-driven host/accelerator partitioning for performance optimization, and at second level a resource-driven sequential/structural partitioning of the accelerator source code to optimize the utilization of its reconfigurable resources. The analysis of candidate (task) performances in CoDe-X has to be done for both, a procedural (sequential) programmable host processor, and the structural programmable data-driven accelerator processor. In complete application time estimation data-dependencies for parallel task execution (host/accelerators) are considered. To stress the significance of this novel application development methodology, the paper first gives an introduction to the Xputer target hardware platform.

1. Introduction

From empirical studies [1] it can be concluded that the major amount in computation time is due to rather simple loop constructs. To speed up the execution of algorithms there exist very often the alternatives of a hardware implementation or of a parallel software implementation. The software solution results in a procedural implementation and requires the development of parallelizing compilers [2], which mainly offer loop transformations for optimization. The code is then executed on a multiprocessor (figure 1). This solution is well-known and can be applied by many application developers. The implementation of the algorithm may be performed at a low development cost compared to the high hardware cost of the general-purpose multiprocessors (figure 1) used to run the algorithms. However, the hardware structures of the execution platform do not reflect the structure of the algorithms very well, which substantially restricts the exploitation of inherent parallelism and acceleration.

The hardware solution results in a structural implementation. Until recently this has been done by describing the algorithm using a (parallel) hardware description language. The description was then transformed into a (hardwired) VLSI-ASIC using design automation tools (figure 1). This results very often in high acceleration factors. Implementing an algorithm on an ASIC is coupled to high development cost at low hardware cost (figure 1) (given a production number is reached). A main disadvantage is that only hardware specialists can follow this way.

But hardware has become „soft“ [3]. Due to the availability of field-programmable logic (FPL) and field-programmable gate arrays (FPGAs) [4], [5], the paradigm of structural programming has been arisen also on the program development level. Accelerators based on such reconfigurable hardware are called custom computing machines (CCMs [6], [7]). Usually they are used in combination with a conventional host computer. Thus, algorithms or parts of algorithms are ‘compiled’ onto reconfigurable hardware. So two different worlds of computation can be distinguished:

- the structural computation (in space on the FPGAs and CCMs), and
- the procedural computation (in time on the host).

Implementing and accelerating an algorithm means now to optimize the hardware/software trade-off, and to reduce the total design time. Hardware/software co-design has introduced a

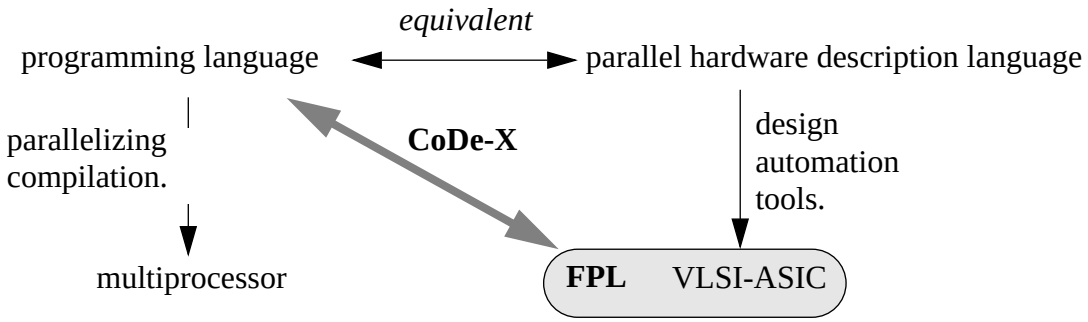

Figure 1. Hardware versus Software Implementation of Algorithms

number of approaches to partition algorithms for the structural and the procedural part under aspects of execution speed-up [8], [9], [10], [11]. This is especially useful for the increasing number of real-time problems and applications.

Currently „structural programming“ requires hardware experts, since (1) contemporary FPGA-based accelerator architectures are far away from being general purpose platforms, and (2) the tools to be used for the description and the code generation are still the hardware developer‘s ones.

A further disadvantage is the fact that the von Neumann paradigm of the host does not efficiently support “soft” hardware because of its extremely tight coupling between instruction sequencer and ALU: architectures fall apart, as soon as the data path is changed. So a new paradigm is desirable like the one of Xputers [12] which conveniently supports “soft” ALUs like the reconfigurable ALU concept (rALU) [12] or the Kress ALU array (reconfigurable data path array) [13], [14], [15].

For such a new class of hardware platforms a new class of compilation tools is needed, incorporating hardware/software codesign techniques and parallelizing compilation methods, in order to link Xputers as universal embedded accelerators to workstations or other hosts. This requirement is met by the CoDe-X compilation framework [16], [17]. The developer is allowed to use a well-known procedural language (here: C) to program his application (figure 1). At a first level, performance-critical parts of an algorithm are identified and migrated to the Xputer without manual interaction. Profile-driven strategies for synchronization overhead reduction, run-time reconfiguration, and memory organization are performed. The second partitioning level creates a resource-parameter-driven procedural/structural partitioning of Xputer source code by generating structural code for e.g. the Kress ALU array, and sequential code for the Xputer’s address generation hardware (the data sequencer hardware [18]).

The goal is to optimize the utilization of the Xputer hardware resources providing parallelism at instruction level. In contrary to hardware/software co-design systems like Cosyma [8] or Vulcan [9], here the problem of accelerators based on a new machine paradigm is discussed. The fundamentally new feature of the CoDe-X framework is the two-level co-design approach.

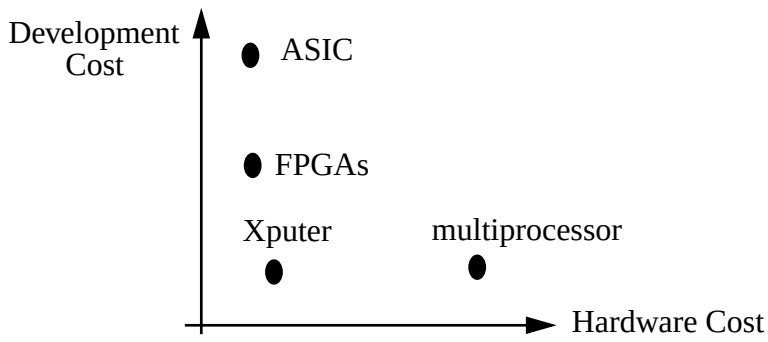

Figure 2. Development Cost versus Hardware Cost

First this paper describes the underlying target hardware in section 2. Section 3 gives a brief overview on the CoDe-X framework and focuses on its performance evaluation process. Then some experimental results are discussed (section 4). Section 5 concludes the paper.

2. The Xputer Paradigm and Architecture

Many applications require iterating the same data manipulations on a large amount of data, e.g. statement blocks in nested loops. The Xputer machine paradigm accelerates such applications. Xputers are especially designed to reduce the von Neumann bottleneck of repetitive decoding and interpreting address and data computations. High performance improvements have been achieved for the class of regular, scientific computations [12], [19], [20].

A Xputer architecture consist of several Xputer modules. The modules are connected to a host computer. Making use of the host simplifies disk access and all other I/O operations. After setup, each Xputer module runs independently from the others and the host computer until a complete task is processed. Each module generates an interrupt to the host when the task is finished. So the host is free to concurrently execute other tasks in-between. This allows the use of the Xputer modules as a general purpose acceleration board for time critical parts in an application.

The basic structure of an Xputer module consists of three major parts (figure 3):

- the data memory
- the reconfigurable arithmetic and logic unit (rALU) including several rALU subnets with multiple scan windows
- the data sequencer (DS) comprising several generic address generators (GAGs)

Depending on the number of modules or the organization of the data sequencer or the rALU several Xputer architectures are possible. In the following the MoM-III Xputer architecture [19], [20] is sketched together with it´s reconfigurable ALU, called the Kress ALU array (KrAA-I) [14].

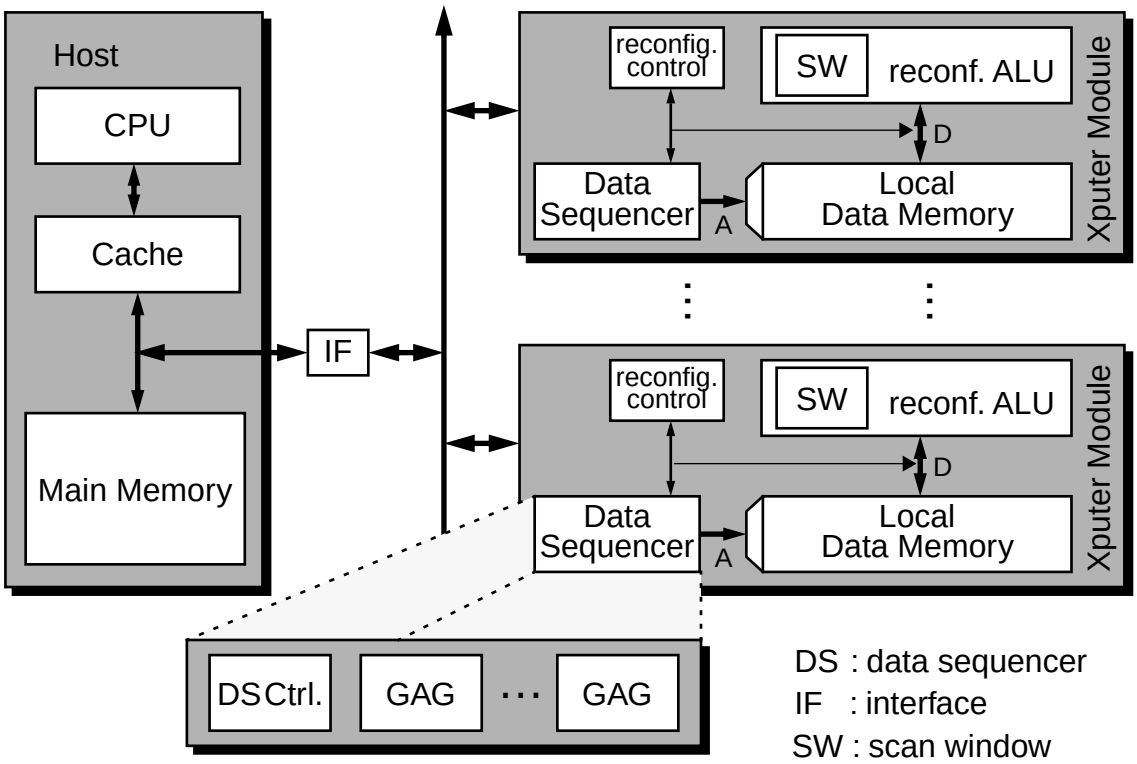

Figure 3. Host with Xputer-based accelerator comprising several modules

2.1 The MoM-III Xputer architecture

The MoM-III accelerator (Map-oriented Machine) is one out of many possible Xputer architectures. It consists of seven modules, connected to a host computer (figure 3).

The data memory which is local on each Xputer module, is primarily organized two-dimensional, but can also be interpreted as higher dimensional. It contains the data which has to be accessed or modified during an application. The data is arranged in a special order to optimize the data access sequences. This arrangement is called data map. The scan windows (SW) serve as interface of the reconfigurable ALU (rALU) subnets to the data memory. A scan window holds a copy of the data out of a local neighbourhood of the data map. Scan windows are adjustable in size during run-time of an application. The main memory of the host and the local memories are logically a single shared memory. The host has access to all local memories, and vice versa. The external bus can be used only by a single Xputer-module at a time.

In accessing arrays and other regular data structures the data sequencer can be specified by a parameter set, from which the reconfigurable generic address generators (GAGs [18]) carry out generic 2-dimensional movement patterns of the scan windows which are called generic scan patterns. Thus GAGs contribute to speed-up by avoiding addressing overhead, since GAGs compute generic scan patterns without needing any memory cycles other than for initial parameter fetch. Thus the data sequencer represents the main control instance of an Xputer. Data sequencing in general means, that the GAGs address data at correct locations in the data memory by generic scan patterns and load it into the scan windows of the rALU subnets. The MoM-III data sequencer is implemented with reconfigurable logic (Xilinx XC4013 FPGA).

All data manipulations are done in the reconfigurable ALU (rALU). To support highly computing-intensive applications we need structurally programmable platforms providing word level parallelism instead of the bit level parallelism of FPGAs. We need *FPAAs* (field-programmable ALU Arrays) instead of FPGAs. For area efficiency this new platform should be suitable for full custom design, as known from ASAPs operating in SIMD mode. But ASAPs are not structurally programmable and support only problems with completely regular data dependencies. What we need is a platform as dense as ASAPs, but structurally programmable. For flexibility to implement applications with highly irregular data dependencies, each PE (processing element) should be programmable individually. Each PE should be a reconfigurable ALU (rALU). Also the local interconnect between particular PEs or rALUs should be programmable individually. Our solution for a FPAA is the Kress Array.

2.2 The Kress ALU Array

The Kress ALU array [14], [15] is a reconfigurable datapath architecture (rDPA). It consists of 72 identical word-oriented datapath units (DPUs) arranged in a two-dimensional structure. The rDPA provides non-multiplexed bidirectional interconnect resources between nearest neighbour DPUs. One DPU of the Kress array presents the following features:

- 32 Bit DPU (reconfigurable Processing Element)
- DPU has rALU, small register file, routing resources
- operations, routing only, routing and operations
- routing-only use substantially adds flexibility
- rALU: arithmetic, relational, logic, special, xfer
- pipe-like asynchronous inter-DPU communication
- smart interface for data scheduling (data streams entering and leaving the FPAA).

In order to have a general purpose interface between the rALU and the GAGs, these components are transport-triggered, which means the availability of data triggers the operation. This allows to be very flexible in implementing different rALU subnets, as the subnets do not need to have the same

computation times. Furthermore, it allows to implement a pipelined rALU concept as well as a combinatorial net. After finishing, the rALU signals the end of the computation to the GAGs.

One major advantage of the Kress ALU Array is its flexibility, e. g. the data paths of the GAGs can be also synthesized into such a device. For more details see [13], [16], [17].

3. The Parallelizing Compilation Framework CoDe-X

For the above hardware platform the parallelizing co-design framework CoDe-X is being implemented which accepts X-C source programs (figure 4). X-C (Xputer-C) is a C extension, including also an optional data procedural language extension [17] [21]. CoDe-X consists of a 1st level partitioner, a GNU C compiler, and an X-C compiler. The X-C source input is partitioned in a first level into a part for execution on the host (host tasks, also permitting dynamic structures and operating system calls) and a part for execution on the Xputer (Xputer tasks). Parts for Xputer execution are expressed in a X-C subset, which lacks only dynamic structures and has some restrictions in the index expressions of array variable subscripts. At second level this input is partitioned by the X-C compiler [22] [23] in a sequential part for the GAGs, and a structural part for the Kress Array (figure 4). Additionally, generic data procedural library functions can be used as C functions within X-C source programs [17], or experienced users may include directly MoPL-code (Map-oriented Programming Language [21]) into the X-C input specification to take full advantage of the high acceleration factors possible by the Xputer paradigm (figure 4).

The first level of the CoDe-X partitioning process is responsible for the decision which tasks should be evaluated on the Xputer-based accelerators and which one on the host. Generally four kinds of tasks can be determined:

- host tasks which contain dynamic structures
- Xputer tasks (candidates for Xputer execution),
- Xputer library functions, and
- MoPL-code segments included in X-C source.

The host tasks have to be evaluated on the host, since they cannot be performed on Xputer-based accelerators. The Xputer library functions and included MoPL-code segments are executed in any case on the Xputer. Thus only their Xputer performance is evaluated, whereas for library functions a list of their performance values is available, and the execution times of MoPL-code segments is determined in using MoPL compiler and DPSS outputs (see equations (1) and (2)). All other tasks

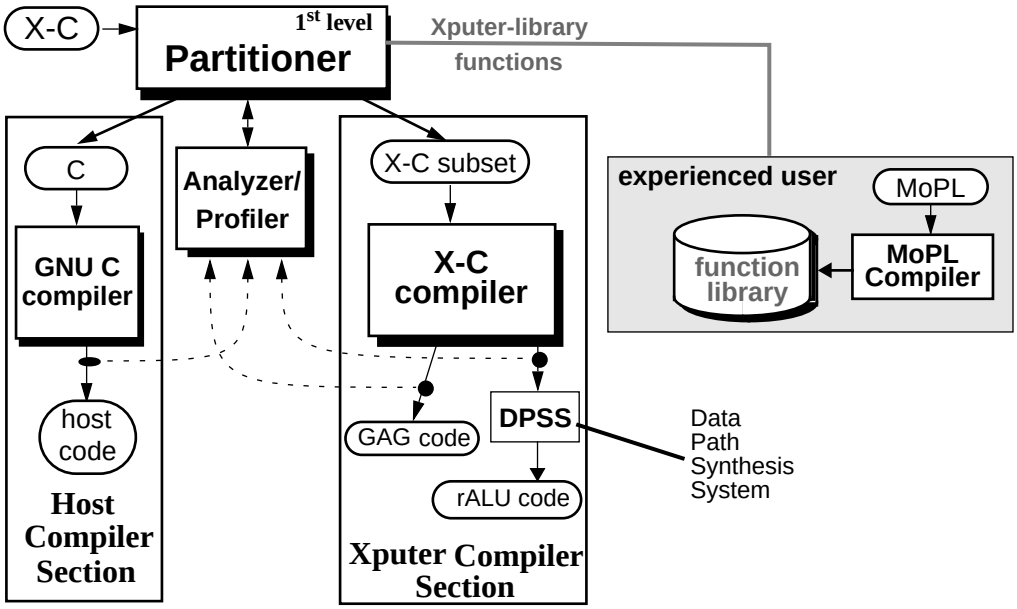

Figure 4. Overview on the CoDe-X Framework

are Xputer tasks, which are the candidates for the first level partitioning process. Due to the program's data dependencies, different possible code optimizations (strip mining, loop fusion, loop splitting, loop unrolling [2]) are applied to these tasks [17], and a performance analysis for host and Xputer execution is done for them. More details about code optimization and partitioning techniques in CoDe-X can be found in [16] and [17]. The paper explains in the following the performance analysis process in the first level host/accelerator partitioner.

The performance values for Xputer execution are determined in using output files from the X-C compiler and the DPSS. During GAG code generation the X-C compiler (figure 4) writes the information how often one compound operator (corresponds to one Kress Array activation) is iterated by one scan pattern into a file. The execution time of such a compound operator (e.g. a loop body within an Xputer task) mapped onto the Kress Array is determined from DPSS generated scheduling diagrams. Such diagrams represent the scheduling of operations inside the loop body and the I/O scheduling of their operands during one Kress Array activation (black parts within plotted example in figure 6). The I/O scheduling determines an optimized sequence of I/O operations through the external bus connecting data memory and Kress Array and the internal on-chip rDPA bus, so that pipelined parallelism of operations during Kress Array activations is maximized by *data scheduling* [14]. From such scheduling diagrams the total cycle number for one activation can be derived. Thus, the Kress Array activation time $t_{ArrayAct}$ can be computed according to equation (1):

$$t_{ArrayAct} = cycle_{num} \cdot \frac{1}{clock_{freq}} \quad (1)$$

whereas:

- $cycle_{num}$: total cycle number for one Kress Array activation, derived from scheduling diagrams.
- $clock_{freq}$: clock frequency of Xputer prototype.

The complete Xputer task execution time t_{exe} is computed by equation (2) (incl. I/O of operands):

$$t_{exe} = num_{ArrayAct} \cdot t_{ArrayAct} \quad (2)$$

whereas:

- $num_{ArrayAct}$: number of Kress Array activations during one scan pattern execution.

The performance values for host execution are estimated by examining and/or profiling the source code of each Xputer task candidate. Therefore three performance tools for sequential processors are integrated into CoDe-X, which are optionally selectable for host performance evaluation:

- the performance anaylsis tool CINDERELLA, developed by Sharad Malik et al. [24].
- the program profiling and tracing tools called "Wisconsin Architectural Research Tool Set" (WARTS) by James Larus et al. [25].
- the program execution time estimator KUKLFLAP [26] (used in CASTLE system [27]).

Malik's tool CINDERELLA is determining a tight bound on a program's *best case* and *worst case excution times* (BCET and WCET). The tool performs a static analysis of the program's executable code, models the instruction cache memory, and computes the bound through the use of Integer Linear Programming. During the analysis, the user can provide additional path information so as to tighten the bound. For each processor type and workstation, another model is required. The behavior of the underlying hardware and operating system has to be deterministic and known. This implies the timing behavior of all hardware components, the effects of caching, pipelining, etc. The operating system must provide static memory management, system calls should have a calculable timing behavior, and no asynchronous interrupts should be allowed [28].

The profiler QPT within the WARTS tool set rewrites a program's executable file by inserting code to record the execution frequency or sequence of every basic block or control-flow edge. From this information the execution costs of procedures in the program can be calculated by considering the execution time of each intsruction [25].

The process of performance estimation in the program execution time estimator KUKLFLAP consists of two steps:

- measuring execution frequencies of C instructions (profiling results)
- determining program execution time by using these profiling results

The first step is instrumenting the Control/Data Flow Graph (by placing frequency counters at loops, ifs, calls, etc.) Thus not the concrete execution path will be stored, but the frequency sums of different program parts. These sums depend on the input data and are stored as compact profile data, whereas the measured counter values are matching the source code easily. The second step is determining the program's execution time. Therefore KUKLFPLAP reads tables of different processor operation costs as well as the measured profiling results, and calculates finally for all availabe processors separately the execution times [26].

An application is represented by a task graph, on which a data dependency analysis based on the GCD-test [2] is carried out. Thus tasks which can be evaluated in parallel can be found. After deriving the host and Xputer performance values of all tasks (incl. code optimized task versions) the overall execution time t_{acc} (equation (3)) of an application using the Xputer as accelerator has to be computed. The time t_{acc} includes delays for synchronization, possible reconfigurations as well as memory (re-) mappings during run time, and considers concurrent host/Xputer task executions. Tasks finally allocated to an Xputer-based accelerator are inputs for the X-C compiler realizing the 2nd level of partitioning in CoDe-X [22] [23]. It performs a paradigm shift from control-procedural von Neumann to data-procedural Xputer, and translates an input task into (possibly vectorized) code which can be executed on the Xputer (figure 4).

$$t_{acc} = \sum_{i \in HT} t_{exe_i} + \sum_{j \in XT} t_{exe_j} + \sum_{j \in XT} t_r + \sum_{j \in XT} t_{mem_j} + \sum_{\forall XputerCalls} t_{syn} - \sum_{j \in XT} t_{ov} \quad (3)$$

whereas:

$\sum_{i \in HT} t_{exe_i}$: sum of task's execution times on the host (HT).

$\sum_{j \in XT} t_{exe_j}$: sum of task's execution times on the Xputer (XT).

$\sum_{j \in XT} t_r$: sum of delays for reconfiguration during run time.

$\sum_{j \in XT} t_{mem_j}$: sum of delays for Xputer data map (re-) mappings during run time (communication overhead).

$\sum_{\forall XputerCalls} t_n$: sum of delays for host/Xputer synchronization.

$\sum_{\forall XT} t_{ov}$: sum of overlapping execution times between concurrently executed tasks.

The delays for run time reconfigurations depends on task's configuration code size, and if partly reconfiguration of the Kress Array can be performed while other parts of the array are active. The times for data sequencer reconfigurations correspond to the re-loading of registers controlling the scan pattern generation, which can be nearly neglected [14].

Memory (re-)mappings are necessary, when two tasks use the same data (data dependent), and the actual data values have to be written into the correct data map locations before starting the dependent task. Therefore, sometimes a different distribution of the data within the two-dimensionally organized memory (re-mapping) is necessary. Thus, the considered delays depend on the sizes of (re-) mapped data portions.

The Xputer Run-Time System (XRTS) [29] provides the software-interface from the host to Xputer-based accelerators, and can be started interactively or inside a host's process. The main purpose of this system is scheduling, controlling and synchronizing applications executed on the accelerator. The corresponding sum of run time delays depends on the number of XRTS activations.

If tasks are not data dependent, they can be executed concurrently, resulting in overlapping task execution time intervals. For each task an ASAP- and ALAP schedule point is computed (relatively to their code positions in the main program), which depend on their data dependencies. This information and the simulated annealing-based task allocation [30] determines the task scheduling, from which these overlapping execution time intervals can be derived. The value of t_{acc} (equation (4)) is used as cost function in each step of the simulated annealing process, and has to be optimized during different task partitions in varying their allocation between host and Xputer-based accelerators.

In the next section some experimental results are given by illustrating the Xputer performance evaluation process of tasks, in viewing an edge detection algorithm.

4. Experimental Results

Edge detection techniques are used primarily as enhancement tools for highlighting edges in an image. Digital filtering with appropriate coefficients is a straightforward spatial-domain technique for edge detection. Given a 996 by 996 pixel grey-scaled image *p* (x,y), a 4 by 4 coefficient-matrix h(i,j). The edge detected image *q* (x,y), is obtained by using equation (4) below, which performs a 2-dimensional non-recursive filter operation:

$$\forall(x,y): \quad q(x,y) = \sum_{i=0}^{3} \sum_{j=0}^{3} p(x+i, y+j)h(i,j) \tag{4}$$

The X-C program of this small example was divided first into four tasks. Two of them were filtered out for being executed on the host in any case. These were tasks containing I/O routines for reading input parameters and routines for plotting the image, which cannot be executed on the Xputer. The remaining two tasks were potential candidates for mapping onto the Xputer (Xputer tasks). In one of these two tasks (nested loop of level 4) loop unrolling was applied by CoDe-X' 1st level partitioner up to the limit of available resources of the Kress Array, resulting in a shorter loop execution time. Additionally, the DPSS is performing different optimizations (loop folding, pipelining in vectorized code etc.) during mapping the task's loop body onto the Kress Array. In the following this task (*task 0*) is taken for illustrating the performance estimation process for Xputer- and host execution. First, the execution time of one Kress Array activation can be derived from DPSS generated scheduling diagrams (for *task 0* see figure 6), as explained in section 3. From this scheduling diagram 345 is the derived number of cycles for one Kress Array activation of *task 0*.Thus, according to the formula in equation (1), the execution time for one array activation is 10.5 µs (33 MHz

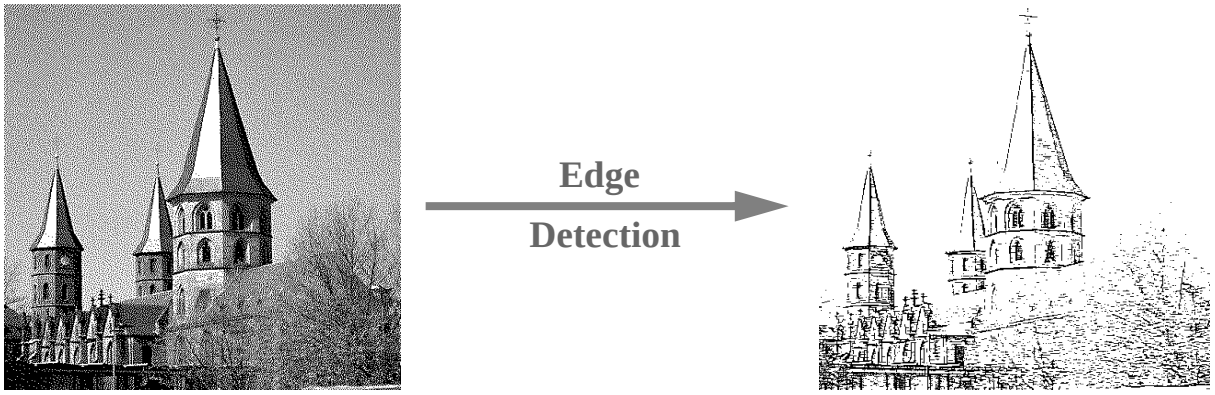

Figure 5. Example of applying edge detection

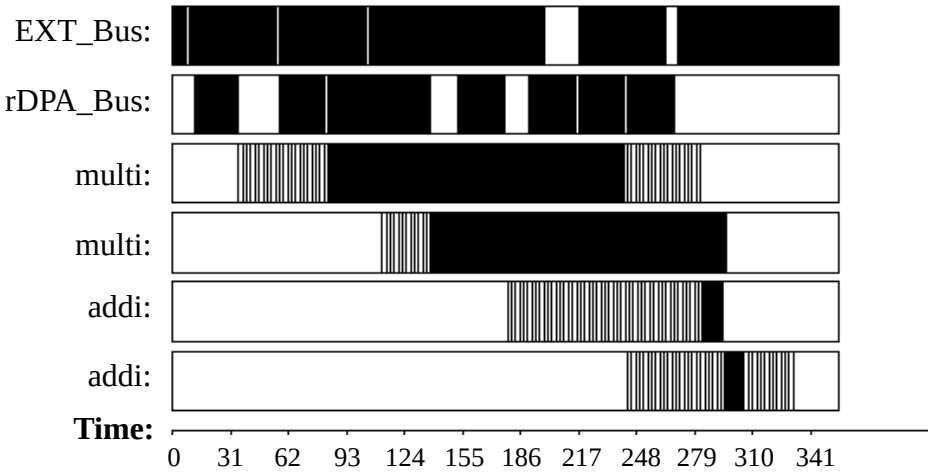

Figure 6. Scheduling diagram of task 0 incl. loop folding

clock frequency assumed). Secondly the number of Kress Array activations can be derived from X-C compiler generated GAG code (see figure 4), which is for *task 0* (996*996), corresponding to the index range of the two outer loops (two inner loops are unrolled). This results in a complete task execution time of 10.42 sec. for *task 0*.

The measured host performance values of *task 0* (for SPARC 10/51) have been approximately 33 sec. by profiler QPT within the WARTS tool set, as well as by KUKLFLAP. Thus, Xputer execution of this task would result in a speed-up of 3. If strip mining would be additionally applied to *task 0* (dividing the image into stripes, which can be manipulated concurrently on different Xputer-based accelerator modules), the speed-up can be increased by a factor 5.

5. Conclusions

The two-level partitioning hardware/software co-design framework CoDe-X and a powerful general purpose reconfigurable hardware platform for software-only accelerator implementation of highly computation-intensive applications have been introduced briefly. The reconfigurable Kress ALU array and the Xputer machine paradigm have been summarized.

CoDe-X accepts C language source programs, and generates sequential code for the host, as well as structural code, data schedules and storage schemes for Xputer-based accelerators. To guide the host/accelerator partitioning process, an extensive task performance evaluation for host and Xputer execution is necessary. Therefore three profiling and performance estimation tools for sequential processors (integrated in CoDe-X) have been sketched briefly. Moreover, the process of task's performance determination for Xputer execution was explained. The obtained task performance values are precise, because for Xputer execution many information must be available at compile time (loop bounds etc.). Additionally, the paper explained the overall execution time estimation of multi-task applications (cost function, to be optimized during partitioning), which has to consider reconfiguration-, communication-, and synchronization-overhead during run time, as well as concurrent task execution of data independent tasks on host and Xputer-based accelerators. Some experimental results were given by analyzing a single task's performance evaluation process for an edge detection algorithm and its Xputer acceleration factors in comparison to a workstation-only version.

References

[1] Z. Shen, Z. Li, P.-C. Yew: An Empirical Study of Fortran Programs for Parallelizing Compiler; IEEE Transactions on Parallel and Distributed Systems, Vol.1, No.3, pp. 356-364, July 1990.

[2] H.P. Zima, B. Chapman: Supercompilers for Parallel and Vector Computers; ACM Press Frontier Series, Addison-Wesley, 1990.

[3] R.W. Hartenstein, J. Becker, R. Kress: Custom Computing Machines vs. Hardware/Software Co-Design: from a globalized point of view; FPL'96, Darmstadt, 1996.

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

[5] R.W. Hartenstein, M. Glesner: Field-Programmable Logic - Smart Applications, New Paradigms and Compilers; Springer Verlag 1996, (Lecture Notes on Computer Science, vol. 1142).

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

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

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

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

[10] K. Buchenrieder: „Hardware/Software Co-Design - An Annotated Bibliography“; IT Press Chicago, 1995.

[11] 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.

[12] R.W. Hartenstein, A.G. Hirschbiel, M. Weber: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware; InfoJapan'90- International Conference memorizing the 30th Anniversary of the Computer Society of Japan, Tokyo, Japan, 1990.

[13] 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.

[14] R. Kress: A fast reconfigurable ALU for Xputers; Ph. D. dissertation, Kaiserslautern University, 1996

[15] R. Kress: Computing with reconfigurable ALU arrays; IT Press (planned for 1997).

[16] R.W. Hartenstein, J. Becker, M. Herz, R. Kress, U. Nageldinger: A Partitioning Programming Environment for a Novel Parallel Architecture; 10th Int’l. Parallel Processing Symposium (IPPS), Honolulu, Hawaii, April 1996.

[17] R.W. Hartenstein, J. Becker: A Two-Level Co-Design Framework for Xputer-based data-driven reconfigurable Accelerators; Proc of the IEEE 1997, HICSS‘30, Wailea, Maui, Hawai, January 1997.

[18] R.W. Hartenstein, H. Reinig: Novel Sequencer Hardware for High-Speed Signal Processing; Workshop on Design Methodologies for Microelectronics, Smolenice Castle, Slovakia, Sept. 11-13, 1995.

[19] R.W. Hartenstein, J. Becker, R. Kress, H. Reinig, K. Schmidt: A Reconfigurable Machine for Applications in Image and Video Compression; Conf. on Compression Techniques & Standards for Image & Video Compression, Amsterdam, Netherlands, March 1995.

[20] R.W. Hartenstein, J. Becker, R. Kress, H. Reinig: High-Performance Computing Using a Reconfigurable Accelerator; CPE Journal, Special Issue of Concurrency: Practice and Experience, John Wiley & Sons Ltd., 1996.

[21] A. Ast, J. Becker, R. Hartenstein, R. Kress, H. Reinig, K. Schmidt: Data-procedural Languages for FPL-based Machines; 4th Int’l. Workshop on Field Progr. Logic & Appl., FPL’94, Prague, Sept. 7-10, 1994, Springer, 1994.

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

[23] K. Schmidt: Xputers: a high performance computing paradigm; IT Press (planned for 1997).

[24] S. Malik, W. Wolf, A. Wolfe, Y.-T. Li, T.-Y. Yen: Performance Analysis of Embedded Systems; NATO/ASI, Tremezzo, Italy, 1995, Kluwer Acad. Publishers, 1995.

[25] J. Larus, T. Ball: Optimally Profiling and Tracing Programs; ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 16, no. 4, pp. 1319-1360, July 1994.

[26] KUKLFLAP program execution time estimator; Gesellschaft f. Mathematik u. Datenverarbeitung, Sankt Augustin , Germany 1995: http://alcatraz.gmd.de:9422/castle/doc/kuklflap.html.

[27] J. Wilberg, R. Camposano, P. G. Plöger, M. Langevin, T. Vierhaus: Cosynthesis in CASTLE; in G. Saucier, A. Mignotte: Novel approaches in Logic and Architecture Synthesis, pp. 355-366, Chapman & Hall, London, 1995.

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

[29] U. Nageldinger: Design and Implementation of a menu-driven Run Time System for the MoM-3; Master Thesis, University of Kaiserslautern, 1995.

[30] R. W. Hartenstein, J. Becker et. al: Two-Level Partitioning of Image Processing Algorithms for the Parallel Map-oriented Machine; 4th Int’l. Workshop on Hardware/Software Co-Design CODES/CASHE ‘96, Pittsburgh, USA, March 1996.