Performance Analysis in CoDe-X Partitioning for Structural Programmable Accelerators

Reiner W. Hartenstein, Jürgen Becker
Universitaet Kaiserslautern
Erwin-Schroedinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, e-mail: hartenst@rhrk.uni-kl.de
URL: http://xputers.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 application development methodology, the paper first gives an introduction to the target hardware platform.

1. Introduction

Due to the emerging development of *field-programmable logic* (FPL) devices in the last years hardware has become soft. Thus, emanating from this technology the new paradigm of *structural programming* has evolved. So, now two programming paradigms are available: *programming in time* and *programming in space*. It is time to become aware of the really existing two kinds of software:

- procedural software (code downloaded to RAM)
- structural software (downloaded to hidden RAM)

Tools like XACT etc. are the code generators for structural software. The R&D area of Custom Computing Machines (CCMs: [1], [2], [3]), such as FPGA-based CCMs (FCCMs [4] [5]), merge both kinds of software to an integrated methodology to speed-up performance. We have obtained a dual software-only implementation: procedural software running on the host, together with structural software running on reconfigurable accelerators. Implementing CCMs has a lot in common with hardware/software co-design to optimize hardware/software trade-off and to reduce total design time [6], [7]. The main difference is the target platform: the structural *program* is frozen into ASICs or other hardware structures. Instead of *structural programming* this is called *hardware synthesis*.

But currently *structural programming* usually requires hardware experts, since contemporary FPGA-based accelerator architectures are far from being general purpose platforms having several severe draw-backs:

- by far too area-inefficient
- designed for random logic only
- too fine grain and too slow
- poor cost/performance ratio for highly computing-intensive (arithmetic) applications

The problem is the wide variety of architectures urged by optimization needs which stem from these draw-backs. Neither in CCMs nor in hardware/software co-design a common model is available. The von Neumann paradigm does not efficiently support "soft" hardware because of its tight coupling between instruction sequencer and ALU [8]. You need a new instruction sequencer, as soon as the data path is changed by structural programming: the architecture falls apart. Thus, a new kind of structurally programmable technology platform is needed. But also a new paradigm is needed like the one of Xputers [13], which conveniently supports *soft ALUs* like in the *rALU Array* concept *(reconfigurable ALU Array)* [9]. Such data-driven Xputer-based accelerators can be integrated as co-processors into state-of-the-art workstations (section 2) for executing appropriate computation-intensive parts of applications. To evaluate hardware/software trade-offs for this target hardware to optimize the complete application execution time, an extensive performance analysis of the input specification is necessary.

The paper is organized as follows: in section 2 structural programmable Xputer-based accelerators are introduced, section 3 gives a brief overview on the CoDe-X framework and focuses on its performance evaluation process. Finally, some experimental results are given in section 4.

2. Xputer-based Accelerators

The Xputer machine paradigm has been introduced and discussed elsewhere [8] [11] [12]. Xputers use one or more data sequencers — in contrast to (von Neumann) Computers, which use an instruction sequencer. 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. A good solution is the Kress Array (see figure 1). It permits to map highly irregular applications onto a regularly structured hardware platform. A subset of the C language is mappable onto the Kress Array by the DPSS (Data Path Synthesis System [9], [10]) subsystem of the CoDe-X framework. The result of this structural programming effort is the configured data path within the Kress ALU Array . The DPSS is a simulated annealing optimizer, which carries out placement and routing [10] of the mapped operations. For more details about the Kress Array (originally called rDPA, reconfigurable Data Path Array) and its instruction level parallelism please see [10], and for the DPSS integration within CoDe-X see [14] [20].

Being procedurally data-driven, the Xputer paradigm is well suited to cooperate with the data-driven Kress Array [14]. The current prototype MoM-III [15] is an Xputer architecture, which

0-8186-7895-X/97 $10.00

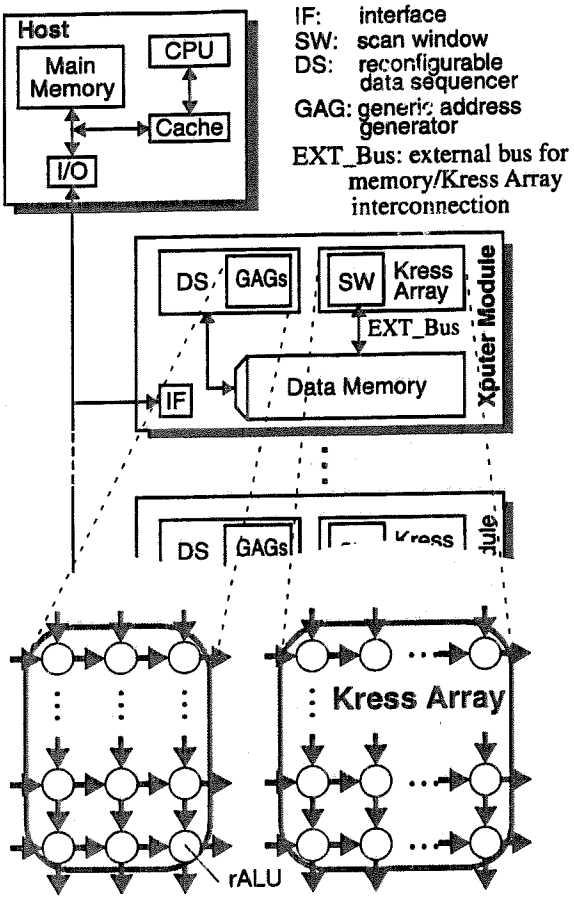

Figure 1. Host with several Xputer modules

consists of several (up to seven) modules (figure 1), connected to a host computer. Making use of the host simplifies disk access and all other I/O operations. After setup, each module runs independently from the others and the host computer until a complete task is processed, and 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 modules as general purpose acceleration boards for time critical parts in an application. A module consists of three major parts [15]:

- primary memory
- reconfigurable Kress ALU Array
- (multiple) generic address generators (GAGs)

A major advantage of the Kress Array is its flexibility, so that also the GAG data paths can be mapped onto ([16], figure 1). So only one major chip design is needed for an Xputer.

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 [8], [11], [14].

In such an environment the parallelizing co-design framework CoDe-X requires 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 Xputer resources. Furthermore, CoDe-X combines three programming paradigms into one more powerful approach: the control procedural paradigm reflected in C language features, the data-procedural paradigm realized in an optional language extension for specifying selected data-procedural application parts executed faster by Xputer hardware [14] [18], and the structural programming paradigm for the reconfigurable Xputer hardware components. Section 3 gives first a brief overview on CoDe-X, and explains afterwards more detailed its performance analysis process for exploring the design space.

3. Performance Analysis in the Dual Co-Design 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 2). X-C (Xputer-C) is a C extension, including also an optional data procedural language extension [14] [18]. CoDe-X consists of a 1st level partitioner, a GNU C compiler, and an X-C compiler (figure 2). 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 [17]. At second level this input is partitioned by the X-C compiler in a sequential part for the GAGs, and a structural part for the Kress Array (figure 1, figure 2). Additionally, generic data procedural library functions can be used as C functions within X-C source programs [14], or experienced users may include directly MoPL-code (Map-oriented Programming Language [18]) into the X-C input specification to take full advantage of the high acceleration factors possible by the Xputer paradigm (figure 2).

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 (2) and (3)). All other tasks 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 [19]) are applied to these tasks [14] [21], and (incl. optimized versions) 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 [14] [16] [20] [21]. 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 2) writes the information how often one compound operator (corresponds to one Kress Array activation) is iterated by one scan pattern into

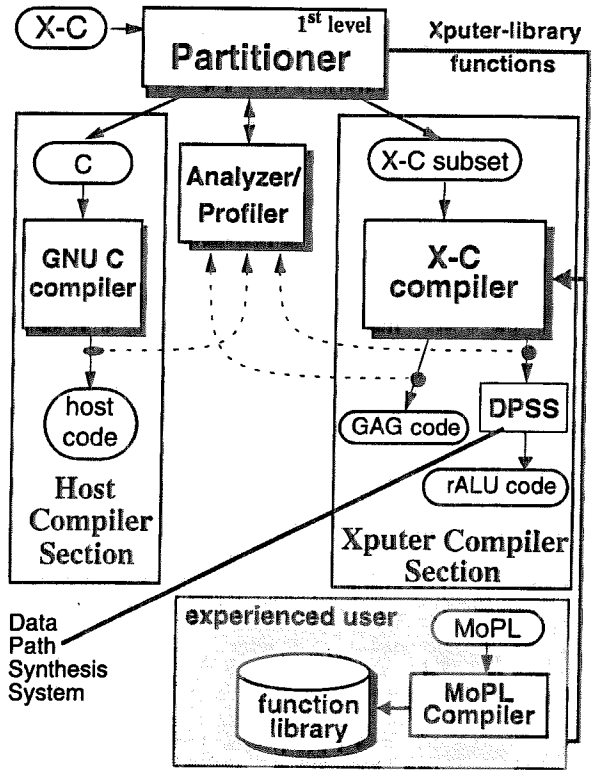

Figure 2. Overview on the CoDe-X Framework

a file. The execution time of such a compound operator (e.g., a loop body within an Xputer task) mapped onto the Kress Array (see section 2) 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 4). The I/O scheduling determines an optimized sequence of I/O operations through the external bus connecting data memory and Kress Array (see figure 1) and the internal on-chip rDPA bus, so that the pipelined parallelism of operations during one Kress Array activation is maximized. For more details about this *data scheduling* see [9] [10]. From such scheduling diagrams the total number of cycles for one activation can be derived. Thus, the Kress Array activation time $t_{ArrayAct}$ can be computed according to equation (2):

$$t_{ArrayAct} = cycle_{num} \cdot \frac{1}{clock_{freq}} \qquad \text{(eq. 2)}$$

whereas:

- $cycle_{num}$: total number of cycles 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 then computed by equation (3) (incl. I/O of operands):

$$t_{exe} = num_{ArrayAct} \cdot t_{ArrayAct} \qquad \text{(eq. 3)}$$

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 different performance tools for sequential processors are integrated into CoDe-X, which can be optionally selected for host performance evaluation:

- the performance anaylsis tool CINDERELLA, developed by Sharad Malik and others [25] [26].
- the program profiling and tracing tools called "Wisconsin Architectural Research Tool Set" (WARTS) by James Larus and others [27] [28].
- the program execution time estimator KUKLFLAP [29] [31] (used in CASTLE co-design system [31] [32]).

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 more details see [25] [26]. 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 [24].

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 [27] [28].

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 [31] [32].

An application is represented by a task graph, on which a data dependency analysis based on the GCD-test [33] 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 (4)) 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. 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 (see figure 2). For more details about the X-C compiler see [9], [17].

$$t_{acc} = \sum_{i \in HT} t_{exe_i} + \sum_{j \in XT} t_{exe_j} + \sum_{j \in XT} t_r + \sum_{j \in XT} t_{mem_j} + \sum_{\forall XputerCalls} t_{syn} - \sum_{j \in XT} t_{ov} \quad \text{(eq. 4)}$$

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 [9] [10].

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. For further details of datamap re-organization during first level partitioning see [23].

The Xputer Run-Time System (XRTS) [22] 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 dependent on their data dependencies. This information and the simulated annealing-based task allocation [20] [21]determines the task scheduling, from which these overlapping execution time intervals can be derived.

The value of t_{acc} (equation (5)) is used as cost function in each step of the simulated annealing process [20] [21] and has to be optimized during different task partitionings in varying their allocation between host and Xputer-based accelerators.

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

4. Example: Experimental Results

Edge detection techniques are used primarily as enhancement tools for highlighting edges in an image [34]. Digital filtering with appropriate coefficients is a straightforward spatial-domain technique for edge detection. Given an N x N image p (x,y), a n by m coefficient-matrix h(i,j). The edge detected image q (x,y), is obtained by using equation (5) below, which performs a 2-dimensional non-recursive filter operation [34]:

$$\forall(x,y): \quad q(x,y) = \sum_{i=0}^{n} \sum_{j=0}^{m} p(x+i, y+j)h(i,j) \quad \text{(eq. 5)}$$

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 hardware resources (here 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 [9] [10]. 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 4), 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 (2), the execution time for one array activation is 10.5 µs (33 MHz clock frequency assumed). Second, as mentioned in section 3, the number of Kress Array activations can be derived from X-C compiler generated GAG code (see figure 2), 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 (CINDERELLA was not used here, because the new processor dependent back-end for SPARC 10/51 was not installed completely). 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 manupulated concurrently on different Xputer-based accelerator modules [20] [21]), 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

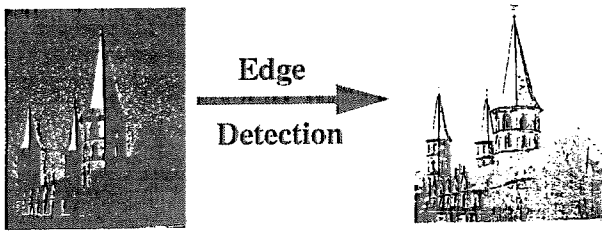

Figure 3. Example of applying edge detection

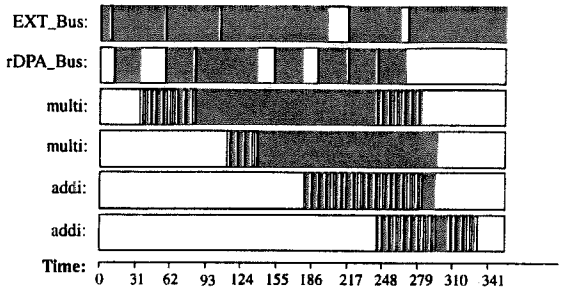

Figure 4. Scheduling diagram of task 0 incl. loop folding

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.

6. References

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

[2] R. Hartenstein, J. Becker et al.: Custom Computing Machines vs. Hardware/Software Co-Design: From a globalized point of view; 6th Int'l. Workshop on Field-Programmable Logic and Applications (FPL'96), Darmstadt 1996

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

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

[5] see [4], but 1994, 1995, and 1996

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

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

[8] R. Hartenstein, A. Hirschbiel, M. Weber: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware; InfoJapan'90- Int'l. Conf. memorizing the 30th Anniv. of Computer Society Japan, Tokyo, Japan, 1990;

[9] R. Hartenstein et al.: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; ASP-DAC'95, Makuhari, Chiba, Japan, Aug. 29 - Sept. 1, 1995

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

[11] R. Hartenstein: FAQ and FQA about Xputers; see http://xputers.informatik.uni-kl.de/

[12] B. Fawcett, J. Watson: Reconfigurable processing with Field Programmable Gate Arrays; Int'l Conf. on Application-specific Systems, Architectures, and Processors (ASAP'96), Chicago, Aug, 1996, IEEE CS Press 1996

[13] R.W. Hartenstein, A.G. Hirschbiel, M. Weber: A Flexible Architecture for Image Processing; Microprocessing and Microprogramming,vol 21, pp 65-72, North-Holland, 1987

[14] Reiner W. Hartenstein, Jürgen Becker: A Two-level Co-Design Framework for data-driven Xputer-based Accelerators; to be publ. in Proc. of 30th Annual Hawaii Int'l. Conf. on System Science (HICSS-30), Jan. 7-10, Wailea, Maui, Hawaii, 1997

[15] R. Hartenstein, J. Becker et al.: A Reconfigurable Machine for Applications in Image and Video Compression; Conf. on Compression Techniques & Standards for Image & Video Compression, Amsterdam, Netherlands, 1995

[16] R. Hartenstein, J. Becker et al.: A Partitioning Programming Environment for a Novel Parallel Architecture; 10th Int'l. Parallel Processing Symposium (IPPS'96), Honolulu, Hawaii, April 1996

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

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

[19] 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

[20] R. Hartenstein, J. Becker et al.: A Parallelizing Programming Environment for Embedded Xputer-based Accelerators; High Performance Computing Symp. '96, Ottawa, Canada, June 1996

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

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

[23] J. Stumpe: Comparison and Reorganizing the storage schemes of variable arrays in the Xputer Multi Tasking Mode; Project Thesis, University of Kaiserslautern, 1996

[24] 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

[25] Y.-T. S. Li, S. Malik: Performance Analysis of Embedded Software Using Implicit Path Enumeration; Proc. of 32nd ACM/IEEE Design Automation Conf. (DAC), pp. 456-461, June 1995.

[26] 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.

[27] 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.

[28] J. Larus: Efficient Program Tracing; IEEE Computer, vol. 26, no. 5, pp. 52-61, May 1993.

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

[30] P. G. Plöger, H. T. Vierhaus: Retargierbare Laufzeitabschaetzungen; to be publ. in Proc. of Workshop Entwurf Integrierter Systeme (E. I. S.) , Univ. Hamburg, 8.-9. April, 1997.

[31] R. Camposano, J. Wilberg: Embedded System Design; J. Design Automation for Embedded Systems, vol. 1, no. 1, pp. 5-50, 1996.

[32] 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.

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

[34] R. C. Gonzalez, P. Wintz: Digital Image Processing; Addison-Wesley Publ. Comp., Inc., Massachusetts, USA, 1977.