A Partitioning Programming Environment for a Novel Parallel Architecture

R. Hartenstein, J. Becker, M. Herz, R. Kress, U. Nageldinger
Universitaet Kaiserslautern
Erwin-Schroedinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, e-mail: abakus@informatik.uni-kl.de

Abstract
The paper presents a partitioning and parallelizing programming environment for a novel parallel architecture. This universal embedded accelerator is based on a reconfigurable datapath hardware. The partitioning and parallelizing programming environment accepts C-programs and carries out both, a profiling-driven host/accelerator partitioning for performance optimization in a first step, and in a second step a resource-driven sequential/structural partitioning of the accelerator source code to optimize the utilization of its reconfigurable resources.

1. Introduction
From empirical studies [23] it can be concluded that the major amount in computation time is due to rather simple loop constructs. Studying DSP and image preprocessing algorithms on a von Neumann platform we found examples where up to 94% of the execution time elapsed for address computation overhead [1]. Parallelizing compilers have been developed, where compilation is based on loop transformations [3], [26]. But the hardware structures are not reflecting the structure of the algorithms very well, which substantially restricts the exploitation of inherent parallelism.

Custom Computing Machines (CCMs: [11], [18]) use application-specific hardware for speed-up. FPGA-based CCMs (FCCMs [4] [5]) use field-programmable hardware as a general purpose accelerator co-processor. But application development here usually requires hardware experts, due to the lack of a suitable general paradigm. The von Neumann paradigm does not efficiently support “soft” hardware because of its extremely tight coupling between instruction sequencer and ALU: processor architectures fall apart, as soon as the data path is changed. So another paradigm is desirable like the one of Xputers, which supports "soft" ALUs like the rALU concept (reconfigurable ALU) [7] or the rDPA approach (reconfigurable data path array) by Rainer Kress [14] [15].

For this new class of hardware platforms a new class of compilers is needed, which generate both, sequential and structural code: i. e. partitioning compilers. This paper introduces a parallelizing compilation method with two levels of partitioning: host/accelerator partitioning and a structural/sequential partitioning (second level).

First the target hardware is briefly summarized. Section 3 explains the partitioning and parallelizing programming environment targeting conventional C programs onto a host using an Xputer as universal configurable accelerator.

2. The Target Hardware
The target hardware consists of a host workstation that uses the Xputer [6] [7] [8] as a universal hardware accelerator (see figure 1). With Xputers the highest speed-up is obtained for algorithms, which iterate the same set of operations over a large amount of data. Such operations are mapped onto a reconfigurable parallel arithmetic-logic unit (rALU). All input and output data to the complex rALU operations is buffered in a scan window (SW), which is a kind of smart register file, the location sequence of which in memory space is determined by a data sequencer. All data in the scan windows can be accessed in parallel by the rALU.

Each of up to seven Xputer modules run independently from each other and the host until a task is completed. Reconfigurations can be performed independently from other running tasks. Each module generates an interrupt to the host when the task is finished. The host may run concurrently to such tasks. The basic structure of an Xputer module consists of three major parts (Figure 1):

- two-dimensionally organized data memory
- reconfigurable arithmetic & logic unit (rALU)
- reconfigurable data sequencer (DS)

Within iterations the access sequence to data, typically organized in arrays, often shows regularity which allows to describe address sequences generically from a few parameters (see section 3.2). The sequence of memory locations (examples in figure 6) reached by such generic address sequences (GAS) we call scan patterns (SP). By this basic sequencing mechanism Xputers are deterministically data-driven (non-von Neumann): i. e. one or multiple (generic address generators) GAs are used instead of an instruction sequencer, generating addresses only by hardware without eating memory cycles. During a SP operation execution is...
transport-triggered. The GAG operates in a two-stage pipeline. The first stage computes the position of the scan window. The second stage computes out of this position the relative addresses of the variables in the scan window (Figure 2).

The data sequencer is implemented with reconfigurable logic (the control logic (Figure 2) by a Xilinx XC4013). rDPA components will be used for GAG datapaths. Thus in addition to the pipeline a fine grained parallelism is achieved. The rDPA is an integrated reconfigurable array [10] [14] [15] of reconfigurable ALU-like 32-bit processing elements (PEs) running in parallel. The rDPA is faster and more area efficient than commercial FPGAs.

The scan pattern generator of the GAG consists of two identical one-dimensional address generators for x- and y-addresses. Each one is realized by three identical address steppers: a Base Stepper, an Address Stepper and a Limit Stepper (Figure 3). One PE in the rDPA array is sufficient to implement one of these steppers (Figure 4). Here the reconfiguration of a GAG is done very fast by re-loading only the registers of the steppers, which represent the parameters of the two scan patterns.

The address stepper steps the address register A from the initial value, which is copied from B or L, by ΔA increments until a final value defined by maximum step count or a threshold given by B, L, F, or C. Most non-trivial scan patterns are nested. That’s why B or L may be used to step through a master scan pattern, whereas A is used to step through a slave scan pattern. Each dimension of a scan pattern is described by seven parameters shown on top of figure 3. A master stepper (B or L) computes within the outer loop the current base address, from which the inner loop steps...
A two-dimensional scan pattern is composed of coordinated parallelism of the two one-dimensional stepper parts, one for the x address $A_x$ and one for the y address $A_y$.

Figure 5 illustrates GAG address generation by a shuffle exchange data transfer (size C-F is 9, destination stepwidth $\Delta A$ is 3, see Table 1) using two 1-by-1 scan windows: SW1 as read-only source and SW2 as write-only data sink. GAG1 steps SW1 through memory space. The shuffle, scan pattern of SW2, is carried out by GAG2. This shuffle is a nested scan pattern example, where the master scan pattern stepping $B$ has $B_0$ or $\Delta B$ as initial location, or, step width, respectively (see Figure 5). The slave scan pattern uses current $B$ (from the master), or $\Delta A$, respectively. Figure 6 shows a few more 2-D scan patterns using a single GAG only. For more details on scan patterns see [2].

### Table 1. Parameters for scan patterns

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameter</th>
<th>Meaning</th>
<th>Values in example</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Floor</td>
<td>Absolute minimum for base</td>
<td>GAG 1, GAG 2</td>
</tr>
<tr>
<td>$\Delta B$</td>
<td>Base step</td>
<td>Step width for base stepper</td>
<td>1, 1</td>
</tr>
<tr>
<td>$B_0$</td>
<td>Initial base</td>
<td>Initial value for base stepper</td>
<td>1, 1</td>
</tr>
<tr>
<td>$\Delta A$</td>
<td>Address step</td>
<td>Step width for address stepper</td>
<td>1, 3</td>
</tr>
<tr>
<td>$L_0$</td>
<td>Initial limit</td>
<td>Initial value for limit stepper</td>
<td>9, 9</td>
</tr>
<tr>
<td>$\Delta L$</td>
<td>Limit step</td>
<td>Step width for limit stepper</td>
<td>0, 0</td>
</tr>
<tr>
<td>C</td>
<td>Ceiling</td>
<td>Absolute maximum for limit</td>
<td>9, 9</td>
</tr>
</tbody>
</table>

For the above hardware platform a partitioning and parallelizing compilation environment CoDe-X is being implemented which accepts X-C source programs (Figure 7). X-C (Xputer-C) is a C dialect. CoDe-X consists of a 1st level partitioner (partially implemented), a GNU C compiler, and an X-C compiler (fully implemented). 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, but includes language features to express scan patterns [2], [20]. At second level this input is partitioned by the X-C compiler in a sequential part for the DS, and a structural part for the rDPA. Experienced users may use special MoPL library functions [2] to take full advantage of the high acceleration factors possible by the Xputer paradigm (see Figure 7).

### 3. The CoDe-X Environment

For the above hardware platform a partitioning and parallelizing compilation environment CoDe-X is being implemented which accepts X-C source programs (Figure 7). X-C (Xputer-C) is a C dialect. CoDe-X consists of a 1st level partitioner (partially implemented), a GNU C compiler, and an X-C compiler (fully implemented). 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, but includes language features to express scan patterns [2], [20]. At second level this input is partitioned by the X-C compiler in a sequential part for the DS, and a structural part for the rDPA. Experienced users may use special MoPL library functions [2] to take full advantage of the high acceleration factors possible by the Xputer paradigm (see Figure 7).

### 3.1 Profiling-driven Partitioning (1st level)

The Xputer tasks are the candidates for the performance analysis on the host and on the Xputer. The user may advise the partitioner about parts to be Xputer-executed in any case. For library functions a list of their performance data is available. First, program parts using MoPL extensions or containing loops which can be transformed by the X-C compiler, are routed for Xputer execution. Such an Xputer job is divided into the following Xputer tasks, which can be computed on a single Xputer module:

- Basic block (a linear code sequence).
- Basic block including conditions.
- Fully nested loops (up to seven loops), or
- Not fully nested loops (up to seven loops).

The restriction to seven nested loops is due to limited resources in the current Xputer prototype. The data sequencer of this prototype contains seven GAGs, which may run in parallel. An Xputer job is represented by a task graph, on which a data dependency analysis based on the GCD-test [25] is carried out. Thus tasks which may run in parallel are found. In this phase parameter-driven transformations are carried out in order to exploit best hardware utilization. For example, single nested loops of large tasks are transformed...
into double nested loops (called strip mining [16]), which permits parallel execution on different Xputer modules. The parameters describe the given hardware resources in order to achieve an optimized performance/area trade-off.

This technique can be used e.g. in image processing applications by dividing an image in stripes of equal sizes in order to manipulate these stripes concurrently [12]. The optimization techniques [16] of loop distribution (splitting of one loop into two loops computing the same), of loop fusion (transforming two adjacent loops into a single loops) and of loop interchanging (switching inner and outer loop) are also performed by the 1st level partitioner in order to exploit potential parallelism and to optimize the utilization of the Xputer hardware resources.

**Profiling.** For each Xputer task candidate, the worst case execution time on the accelerator as well as on the host is computed. The performance data for the accelerator is received from the X-C compiler by analyzing the execution time of the rALU operators and their iterations [10], [12].

Host performance is approximated by code examination. For each host type, another model is required. The timing behavior of the underlying hardware and operating system has to be deterministic and known. The operating system must provide static memory management, system calls should have a computable timing behavior, and no asynchronous interrupts should be allowed [19]. The techniques used are similar to these from Malik et al. [17].

The profiler is also computing the overall execution time during run time (see equation (1)).

\[
t_{acc} = \sum_{i \in HT} t_{exe_i} + \sum_{j \in XT} t_{exe_j} + \sum_{j \in XT} t_{r} + \sum_{j \in XT} t_{mem_j} + \sum_{\forall XputerCalls} t_{syn} - \sum_{j \in XT} t_{ov}
\]

whereas:

- \( t_{exe_i} \): sum of execution times of tasks executed on the host (HT)
- \( t_{exe_j} \): sum of execution times of tasks executed on the Xputer (XT)
- \( t_{r} \): sum of delay times for reconfiguring the Xputer during run time
- \( t_{mem_j} \): sum of delay times for re-mapping the 2-dimensional organized Xputer data map during run time
- \( \sum_{\forall XputerCalls} t_{syn} \): sum of delay times for synchronizing host/Xputer (operating system calls)
- \( \sum_{\forall XT} t_{ov} \): sum of overlapping execution times between tasks executed simultaneously on host/Xputer

The overall execution time is determined by delay times of the tasks (host and Xputer), plus the above mentioned penalty times (see \( t_{acc} \), equation (1)). Memory re-mappings are necessary, when two tasks use the same data onto the Xputer, but they are needing a different distribution of the data within the two-dimensional organized memory. Then a re-ordering of the data items must be performed before the execution of the second task starts, which must be also considered in equation (1). The value of \( t_{acc} \) is used as COST-function in the simulated annealing process [22], [12] and has to be optimized by computing different partitionings of Xputer tasks in varying their task allocation between host and Xputer.

### 3.2 Resource-driven Partitioning (2nd level)

The X-C compiler realizes the 2nd level of partitioning and translates an X-C program into code which can be executed on the Xputer. The compiler performs a data and control flow analysis. First the control flow of the program graph is partitioned according to the algorithm of Tarjan [24] resulting in a partial execution order. This algorithm is partitioning the program graph into subgraphs, which are called *Strongly Connected Components*. These components correspond to connected statement sequences like fully nested loops for example, which are possibly parallelizable. The Xputer
hardware resources are exploited in an optimized way by analyzing, if the structure of statement sequences can be mapped well to the Xputer hardware avoiding reconfigurations or idle hardware resources [20]. Xputers provide best parallelism at statement or expression level. So we try to vectorize the statement blocks in nested for-loops according to the vectorization algorithm of J. R. Allen and K. Kennedy [13], after a data dependency analysis has been performed [25].

The access sequences of up to 4 fully nested loops containing variables with linear index functions can be handled by transforming them into parameters for configuring the GAGs (see figure 1). A configuration file for the X-C compiler gives the current restrictions of the hardware, e.g. number of GAGs available, size of rALU subnets etc., which allows a flexible handling of the compiler for future up-grades of the Xputer hardware. Further details and examples about the X-C compiler can be found in [10], [20], [21].

4. Conclusions

A partitioning and parallelizing programming environment for Xputers has been presented. The Xputer is used as universal accelerator hardware based on reconfigurable datapaths. One of the new features is the two level partitioning approach. C-programs are accepted and the profiling-driven host/accelerator partitioning for performance optimization as well as the resource-parameter-driven sequential/structural partitioning for accelerator programming are carried out. At first level, performance critical parts of an algorithm are localized and shifted to the Xputer without manual interaction. The second level carries out a resource-driven compilation/synthesis from accelerator source code to optimize the utilization of its reconfigurable datapath resources. From several application examples encouraging acceleration factors have been obtained in comparison to a single workstation without an accelerator-board [12].

5. References