State-of-the-art Reconfigurable Multithreading Architectures

Pavel G. Zaykov, Georgi K. Kuzmanov, Georgi N. Gaydadjiev
Computer Engineering Laboratory,
EEMCS, TU Delft, P.O. Box 5031, 2600 GA Delft, The Netherlands
{P. Zaykov, G.K.Kuzmanov, G.N.Gaydadjiev} ∋ tudelft·nl

Technical Report: CE-TR-2009-02

Abstract

This report provides a survey on the existing proposals in the field of reconfigurable multithreading architectures (ρMT). Until now, the reconfigurable architectures have been classified according to implementation or architectural criteria, but never based on their ρMT capabilities. More specifically, we identify reconfigurable architectures that provide *implicit*, *explicit* or *no architectural support* for ρMT. Further subdivision of these three classes is also provided by the proposed taxonomy. For each of the proposals, we discuss the conceptual model, the limitations and the typical application domains. We also summarize the main design problems and identify some key research questions related to highly efficient ρMT support. In addition, we discuss the application prospectives and propose possible research directions for future investigations.

1 Introduction

Contemporary embedded systems require high processing power and often employ varieties of different functionalities. Modern appliances, such as digital cameras, mobile phones, personal media players, handheld gaming consoles and many electronic devices used in medical and automotive industries employ multithreading supported either by the Operating System (OS) or by dedicated hardware mechanisms. In these devices, threads are typically employed for processing(exchanging) data among multiple sources. Such examples are: drivers that handles user interaction from touchscreen displays/ buttons, management of wireless network protocols, multimedia (audio and video) computations, etc. During the quest of maximum performance and flexibility, the hybrid architectures combining one or more embedded General Purpose Processors (GPPs) with reconfigurable logic have emerged. There is a clear trend which shows that in the near future there will be more embedded systems integrating recon-

figurable technology. The first indications of such approaches were presented in [51], [21], [39]. It is envisioned that multithreading support will become an important property of such systems.
One of the fundamental problems in multithreaded architectures is efficient system resource management. This has been successfully solved in contemporary GPPs using various implicit and explicit methods. In literature [50], the explicit techniques have been further partitioned into three main categories: Block Multithreading (BMT) - employing Operating System (OS)/ compiler approaches and Interleaved/ Simultaneous Multithreading (IMT/ SMT) using hardware techniques. However, none of these solutions can be applied straightforwardly for managing reconfigurable hardware resources. The main reason is that the reconfigurable hardware is changing its behavior per application, unlike the GPPs, which have fixed hardware organization regardless the programs running on them. Yet, current state-of-the-art architectures do not provide efficient holistic solutions for accelerating multithreaded applications by reconfigurable hardware.
In this technical report we approach the reconfigurable multithreading (ρMT) architectural problems both from the hardware and the software prospective. The specific contributions of the report are as follows:

- We analyze a number of existing reconfigurable proposals with respect to their architectural support of ρMT. Based on this analysis, we propose a taxonomy with three main classes, namely: reconfigurable architectures with *explicit*, *implicit* and *no* ρMT *support*. Further subdivision of these three classes is also provided;

- We summarize several design problems and state open research questions addressing performance efficient management, mapping, sharing, scheduling and execution of threads on reconfigurable hardware resources;

- We provide our vision for promising research directions and possible solutions of the identified design problems;

The technical report is organized as follows: in Section 2, a taxonomy covering related projects is presented. More details about the design problems and the status of the current state-of-art, completed with our vision on some possible application prospectives and potential research directions are described in Section 3. Finally, the concluding remarks are presented in Section 4.

2 A Taxonomy Of Existing Proposals

A taxonomy on Custom Computing Machines (CCM) with respect to explicit configuration instructions has been already proposed in [40]. However, that study did not consider multithreading support as a distinguishing feature. In this section, we introduce a taxonomy of existing reconfigurable architectures with respect to the ρMT support they provide. We identify three main classes

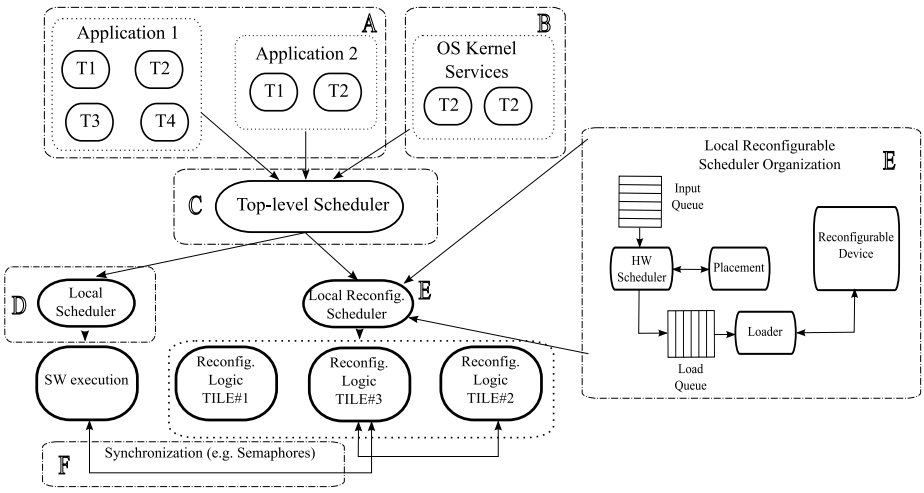

Figure 1: A Conceptual Behavioral Model of ρMT Related Projects

of such architectures, namely: with *explicit*, with *implicit*, and with *no architectural* ρMT support. Note, the meaning that we relate to the definitions of *explicit* and *implicit* ρMT, is different from what is used in GPP systems. In general purpose systems, the classification is based on multithreading support from algorithmic point of view [50]. In our taxonomy we use as a distinguishing feature the presence of architectural/ μ-architectural extensions for creation/ termination of multiple threads on reconfigurable logic. If we classify the ρMT research projects based on the GPP explicit multithreading technique, our taxonomy would look like as follows:

– Reconfigurable Block Multithreading (ρBMT): e.g. [51], [58], [59];
– Reconfigurable Interleaved Multithreading (ρIMT): e.g. [27];
– Reconfigurable Simultaneous Multithreading (ρSMT): e.g. [48], [35];

In this technical report, we consider a different classification prospective. In architectures with *no* ρMT *support*, application threads are mapped into reconfigurable hardware using software techniques – either at the OS or at the compiler level. This software approach provides unlimited flexibility, but the performance overhead too often penalizes the overall execution time especially for real-time implementations. On the other hand, architectures with *implicit* ρMT support, provide performance efficient solutions at the cost of almost no flexibility due to the fixed underlying microarchitecture (μ-architecture) facilitating multithreading. To exploit the flexibility provided at both the software level, as well as by the reconfigurable hardware at the μ-architectural level and to achieve higher system performance, a third emerging class of architectures is identified and termed as architectures with *explicit* ρMT support. Hereafter, we enlighten the proposed taxonomy through examples of existing reconfigurable architectures.

A conceptual behavioral model of an ρMT system is depictured in Figure 1. The

picture represents the basic steps in the management and execution process of multiple threads. Initially, the programmer creates applications (tasks – Section A) or kernel service (Section B) composed of multiple threads. Later, during run-time when an application is selected for execution, depending on the system status information, the Top-level Scheduler (Section C) passes threads to local schedulers (Section D and E). The local reconfigurable scheduler (Section E) accommodates multiple units - queues, scheduling algorithm, placement technique and loading process. The synchronization between different threads is managed by e.g. semaphores (Section F). The different sections of the behavior model, depicted in Figure 1, are implemented either at the software level, or at the μ-architectural level, depending on the particular architecture. Hereafter, we shall reveal how different popular reconfigurable proposals manage the scheme from Figure 1 and based on their architectural support for ρMT, we shall classify them.

2.1 Modern state-of-the-art reconfigurable architectures

The reconfigurable hardware allows the designer to extend the processor functionality both statically and at run-time to speed up the application by executing its critical parts in hardware. In [14], a survey on architectural proposals targeting GPP cores extended with reconfigurable logic is presented. However, that paper has not considered ρMT as a classification criterion. In the years after, a few more reconfigurable proposals have been introduced, capable to be supported by an OS without any specific hardware modifications. We choose to briefly introduce the following two of these later reconfigurable projects, uncovered by [14], because we consider them as a natural evolution of contemporary embedded systems and potentially good candidates for future explicit ρMT extensions:

MOLEN: We choose The Molen Polymorphic Processor [51] proposed by CE Lab, TUDelft, The Netherlands, as an example of tightly coupled (processor/ co-processor) fine-grained reconfigurable architecture. It combines a GPP with several reconfigurable Custom Computing Units (CCU-es). The processor has an arbiter, which partially decodes and issues instructions to either of the GPP or the reconfigurable coprocessor (RP). A general one-time extension of the instruction set is proposed to support an arbitrary functionality implemented in the CCU. The very basic operations of the RP are "set" and "execute". The "set" instruction configures the CCU for a particular functionality and the "execute" instruction performs the actual computation on the CCU. Altogether up to eight instructions are proposed to extend the instruction set to achieve the complete system functionality. The Molen concept has been successfully employed in various projects in the domain of multimedia [23] and cryptography [12]. In the Molen original papers, multithreading has not been discussed, but a follow-up research towards multithreading has been reported in [48]. An overview of this enhanced MT version of Molen is examined in the Subsection 2.3.

Montium TP: As an example of a Coarse Grained Reconfigurable Array

(CGRA) processor core, we choose Montium TP [21], designed by RECORE Systems. It combines five identical Custom Reconfigurable Processing Units (CRPU) in a single chip, connected through a simple 2D-mesh communication infrastructure. This architecture has the following characteristics: once configured, it does not issue any instructions (just processes the data). It does not have a fixed instruction set architecture (ISA) - the application is encoded at microcode level and has fast reconfiguration response time, because of its coarse-grained hardware structure. In its current implementation, the Montium TP is capable to support execution of multiple threads (applications) but only at the OS level. The CRPU does not have support for context-switching - multiple threads are not capable to share the same hardware resources. The processor was originally targeting the domain of streaming applications: i.e., broadcast and multimedia.
Although, in our investigations, we are particularly interested in the embedded systems domain but for the completeness of our survey, we also briefly consider a few large-scale (high-performance) proposals. Many contemporary mainframe/ supercomputing platforms, such as Altix Family by SGI [3], ProLiant server by HP [2], Convey hybrid-core HC-1 [4], RAMP emulation platform [1], employ reconfigurable hardware to speed-up computationally intensive kernels. Though no particular attention on ρMT on these machines is paid in the literature, we believe that multithreading applications can be mapped on them using different (no trivial) software techniques [47].
Preliminary investigations indicate that an efficient ρMT processor would allow better overall system performance [48] as the thread management overhead at the software level could be dramatically reduced. In addition, [48] provides clear indications that multithreading should be also addressed by the state-of-the-art real-time reconfigurable systems at the hardware level.

2.2 Architectures with no ρMT support

As we have already classified architectures with no ρMT support provides simultaneous execution of multiple threads at the software level only – either by the OS or by the compiler without any explicit support.

2.2.1 OS support for ρMT

In this section, we group all known OS targeting reconfigurable devices and implementing in software - Section A, B, C, D, E and F from Figure 1. The first proposal, which identifies some of the necessary services, that an Operating System for reconfigurable devices should support, is presented in [58] and [16] by a research group at the University of South Australia.
BORPH [42]: The research work presented in [42] and [43] by the University of California - Berkeley, identifies that application migration from one reconfigurable computing platform to another, using conventional codesign methodologies, requires from the designer to learn a new language and APIs, to get familiar with new design environments and re-implement existing designs. Therefore,

BORPH is introduced as an OS designed specifically for reconfigurable computers, sharing the same UNIX interface among hardware and software threads, which speeds up the design process. The major difference between BORPH and conventional OS-es for Field Programmable Gate Array (FPGA) architectures comes from the fact that the system reconfigurable logic are treated as a first-class computational resources instead of coprocessors. The BORPH contains three basic components: concept of hardware process and a set of universal interfaces - input/ output registers (IOREG) and hardware file input/ output (I/O) interface. The proposal has the following limitations - hardware threads are executed on but do not share reconfigurable resources. Experimental results are produced from simple applications such as: wireless signal processing, low density parity check decoder and MPEG-2 decoding.
SHUM-uCOS [60]: Another design, tackling the problems caused by the essential differences between software and hardware-tasks is the SHUM-uCOS by the Fudan University, China [60]. The authors propose an real-time OS (RTOS) for reconfigurable systems employing uniform multi-task model. It traces and manages the utilization of reconfigurable resources, improves the utilization and the parallelism of the tasks with hardware task preconfiguration. Detailed descriptions of the abstract layers and their functionality are presented in [60] and [61]. For evaluation of the system, the authors use benchmarks and multiple voice over Internet protocol (VOIP) compression/ decompression algorithms.

2.2.2 Compiler techniques for multithreading on reconfigurable platforms

The most common feature of the architectures grouped in this subcategory is the responsibility of the compiler for task partitioning, scheduling and management of the system resources. The major reason to employ multithreading in these architectures is to hide reconfiguration latencies.
MT-ADRES [59]: In MT-ADRES by IMEC, Belgium, the DRESC Compiler Framework [30] has been extended to support several threads.
The most significant limitation of this proposal is the inability to execute/ terminate threads at run-time which is posed by the compiler static scheduling and optimization algorithms, operating with Control Data Flow Graph (CDFG). Control decisions, such as hiding the reconfiguration latencies and resource management are taken at compile time. Due to the implementation complexity and the fact that the Very Long Instruction Word (VLIW) processor and the CGRA have complete access to the register file, the DRESC compiler [30] limits the execution to only one computing resource at a time, which reduce potential performance gains. All experiments providing information about MT-ADRES performance are achieved through multimedia simulations.
UltraSONIC [20]: Another proposal falling in this category is the UltraSONIC project, represented by Sony Research Labs, UK [20]. It is a reconfigurable architecture optimized for video processing. It has a list of Plug-In Processing Elements (PIPEs), connected through several buses. The programmer receives an architecture abstraction through an API interface. In [57] and [31], the au-

thors introduce multitasking to the architecture. The goal is achieved through two-phase clustering algorithm working on a Directed Acyclic Graph (DAG). The phases are: partitioning (based on Tabu Search) and list scheduling (a static technique). The algorithm places and schedules tasks, applying the following system constraints: FPGA resources, shared resource conflicts, configuration time, communication and processing overhead. The system also has a Task Manager, responsible for task creation and termination procedures. Because of its static nature, the architecture has the same limitations as the MT-ADRES project [59]. The system is initially designed for the multimedia and the data encryption application domains.

2.3 Architectures with implicit ρMT support

The proposals from this category share one common feature - the detailed multithreading support on reconfigurable threads is *implicit*, i.e. hidden from the system programmer. The Instruction Set Architecture (ISA) does not have dedicated special instructions for thread creation and termination procedures. The functionality is achieved with μ-architectural extensions while preserving the architectural model. Bellow, we describe some of the existing proposals falling into the category.

Reconfigurable Extensions for the CarCore Processor: In [48], the authors combine a simultaneous multithreaded processor (SMT) CarCore [49](a simulation model, architecturally compatible with Infenion TriCore 1 Processor) with a Molen style reconfigurable coprocessor [52]. To minimize the complexity of the implementation, the authors employ several constrains to the architecture. They modeled a hardware scheduler, which supports execution on reconfigurable logic of only one thread at a time, preserving the real-time capability for it. Once a thread is started for hardware execution, it could not be interrupted until it is finished (no context-switching). There is no additional ISA extensions for reconfigurable thread management. The ISA extension comprises only the Molen polymorphic ISA and no additional specific instructions for multithreaded support. Meanwhile, other non-real-time threads can continue their execution employing the latencies of the real-time thread. The implementation includes two scheduling policies – fixed-priority and round-robin, over four executing threads.

The **REDEFINE** project [38], [7] by the Indian Institute of Science, Bangalore, proposes a synthesis methodology to realize applications written in a high level language (HHL) on the coarse-grained Runtime Reconfigurable Hardware (RRH). Contrary to related projects in the field, they assume that the whole application could be represented as a set of custom instructions executed on RRH. The custom instructions are not based on occurrence statistics, but are based on co-execution (e.g. both paths of branch instruction). The transformation of the application (in HHL) to hardware proceeds in three steps: 1) The compiler transforms HHL specification to DFG; 2) The compiler partitions the DFG into various application substructures called HyperOps (equal to a thread in our classification); 3) The HyperOps are synthesized into hardware config-

urations. At run-time, depending on the availability of computing elements on the fabric and the data dependencies among different HyperOps, a subset of clusters (composed of HyperOps) ready for execution are scheduled based on pre-order depth first search. Each HyperOp contains information about its data-dependent successors. Because of the lack of detailed description, we assume that this control information is provided implicitly.

Hthreads [35]: The Hthreads(Hybrid Threads) model presented by University of Kansas [35], [8] is multi-layer computational architecture which aims to bridge the gap between the programmers and complex reconfigurable devices. Some of the main system features are the migration of thread management, synchronization primitives and run-time scheduling services (Figure 1, Section F) for both hardware and software threads into hardware module accessed from the GPP only through an universal bus. The authors represent hardware threads with user defined component (designed by the programmer), state controller and universal interface (Register Set). Synchronization procedures are performed through semaphores. In the proposal, the CPU is only interrupted when a change in the system state requires the CPU to switch to another activity. Such changes include timers expiring, devices completing an assigned activity and generating an interrupt. The basic system components are: 1) software thread management (SWTM) which is only responsible for scheduling of software threads. It is executed in parallel with the CPU threads, which reduce the overhead and context switching jitter. The SWTM scheduler manages all CPU interrupt requests, including external-device interrupt, expiring timers, terminating, blocking and unblocking threads; 2) hardware thread interface component (HWTI) which provides management and distributed control (through command and status registers) of threads executed on reconfigurable resources. Some of the major limitations in the implementation of the Hthreads model are as follows: 1) threads executed on reconfigurable resources are not scheduled, instead they are directly loaded when it is necessary and 2) threads are not sharing the reconfigurable resources even when the thread is marked as blocked/ idle. Because of the fact that the system does not have modifications at architectural and μ-architectural levels, the proposal is classified as an *implicit* ρMT. The experimental results are provided in the image processing application domain.

Reconfigurable Multithreaded Architecture Model [56], [55]: The proposal is presented by a research group in the Hamburg University of Technology, Germany. Their primary idea is to map computational threads via pipelined configuration technique into available physical reconfigurable hardware resources. The fixed resource limitations are overcome by virtualizing the computational, communication and memory resources in the reconfigurable hardware. The architecture is based on a synchronous multifunctional pipeline flow model using coarse-grained reconfigurable processing cells and reconfigurable data paths. Descriptors are used for run-time and partial reconfiguration, which enables the processor cells to be configured by Time Division Multiple Access (TDMA). By itself, the descriptors represent small configuration templates in special opcodes, extending a conventional ISA. Therefore,

the ISA grows proportionally with the design complexity and the number of the configuration templates. Compared to existing architectural proposals, the difference comes from the fact that, the authors do not employ the GPP to control the reconfigurable resources. Instead, a hardware approach is taken - a Microtask controller is employed. Current implementation does not support dynamic (runtime) scheduling of incoming workloads. The ideas are not applied in heterogeneous systems yet, represented by a combination of GPP and reconfigurable logic. The designer is responsible for partitioning and mapping the CDFG in microtasks (subtasks), by allocating the flow graph nodes to the system processing resources. The simulation results of streaming multimedia applications are studied.
The research group at the university of Karlsruhe [10] introduces an architecture capable to manage execution of multiple run-time threads (called Special Instructions - SI) through a 'Special Instruction Scheduler'(SI scheduler). Each Molecule is composed of one or several Atoms representing elementary data paths. Multiple Molecules (varying in resource usage & performance) compose each Special Instruction. As a result, the SI Scheduler implicitly selects (without additional control instructions) for execution a mixture of dynamically loaded data paths with conjunction with base processor instructions. The authors examines multiple run-time algorithms based on Molecule loading sequences. Because of the reduced granularity and increased possibility of resource reusage, the system achieves high system performance, tested with H.264 and CIF-video applications.

2.4 Architectures with explicit ρMT support

The basic idea of this ρMT class is to combine the flexibility of the software and the reconfigurable hardware with the potential performance efficiency of the latter and to support ρMT, both at the software level and at the μ-architectural level. There are several partial solutions in the literature which do not provide such a compete mixed model of ρMT - the software and the hardware corporate together to provide simultaneous execution of multiple threads. In such a model, the system services (e.g. scheduling, resource management) should be optimally separated between software and μ-architectural levels. Combined with efficient memory management and thread/function parameters exchange through dedicated registers, an architecture with *explicit* ρMT support would potentially reduce the intra- and inter- thread communication costs. Similar approaches are taken in the following proposals:
OS4RS [29]: In [29], [32] and [33], a research group at IMEC, Belgium, investigates the concepts and reveals some of the open questions, raised by the run-time multithreading and interconnection networks [28] for heterogeneous reconfigurable SoC. The novelty of their approach resides in the integration of the reconfigurable hardware in a multiprocessor system completely managed by the OS for Reconfigurable Systems (OS4RS). The system maintains several threads by a two-level scheduler. The high-level scheduler is handled in software by the main GPP, which stores the running tasks as a linked list. The low-level/

local scheduler can be implemented in software or hardware depending on the type of the slave computing resources (GPPs or reconfigurable logic). Note, that in the current implementation of OS4RS, hardware threads are not sharing the same reconfigurable resources. The OS has several services executed on the main GPP, responsible for monitoring the status of the heterogeneous system and distributing the workload among slave processing units. Due to the fact that a software approach is taken to solve heavily computational problems, such as real-time scheduling, resource allocation and loading, it will eventually become a system bottleneck during heavy computation periods. In their current implementation, the top-level scheduler (Figure 1, Section C) is implemented in software and the local-level hardware (reconfigurable) scheduler (Figure 1, Section E) is not implemented, yet. The authors also propose a proof-of-concept method for context-switching and migration between heterogeneous resources by saving the task state. The questions related to thread state translation between GPP register set and reconfigurable logic are still open. The OS4RS has been tested in JPEG frame decoding and experimental 3D video game. According to the project time schedule, the next generation of the system is expected to be designed between 2008 and 2010.

Reconfigurable Multithreaded processor [26], [27] by the University of Wisconsin-Madison: The authors augment SandBlaster 3000 simulator [19] with Polymorphic Hardware Accelerators (PHAs), which combine properties of functional units and reconfigurable hardware. The processor by itself has four multithreaded Digital Signal Processor (DSP) cores and an ARM processor that provides support for user interface and OS. The research study investigates potential benefits of closely coupled reconfigurable hardware to multithreaded processor. The work could be separated into two topics: the first one is to investigate architectural techniques to provide hardware-software interface between the multithreaded processor and PHAs, the second one is to evaluate the potential benefits of incorporating PHAs in a multithreaded DSP to improve the system performance. The PHAs are implemented as a functional units at the execution stage of the processor pipeline. The instruction set is extended with four instructions for read/ modify the PHA state/ mapping procedures. The multithreading is mainly employed to hide the reconfiguration time. Each one of the PHA blocks contains PHA control interface, reconfigurable block(executing user logic) and optional registers. Configuration of the PHA is done, by loading a sequence of instructions to specific register, accessed by PHA control interface. If a certain high priority task requires a PHA, it is only dedicated to it, without any interference with the other threads. In case of interrupt, thread's PHA inner state could be saved and the unit is released. Therefore, the processor supports context switching over reconfigurable resources. In case of a lack of PHAs, a realtime thread could preempt a non-realtime one. Once configured, in case of identical PHA instructions, the PHA could be reused by different threads. Because of the fact that PHAs are not sharing the same reconfigurable area, there is no necessity for placement algorithm. Some of the system model assumptions are: the PHAs are not sharing the same reconfigurable resource area, as a result there is no necessity for placement algorithms. The architecture is limited to In-

terleaved Multithreading called Token Triggered Threading. The authors argue the choice of such an approach instead of Simultaneous Multithreading, because of the possible power consumption reduction. The authors investigate two PHA binding techniques - static & dynamic. The implementation includes only static (compile time) mapping approach. In case of a run-time binding, the system provides realtime constraints by restricting PHA reusage among threads.

2.5 Summary of the Proposed Taxonomy

Based on the criteria of the provided ρMT support, the aforementioned architectures can be briefly classified as follows:
I. *No architectural ρMT support:*
I.1. OS support for ρMT: Molen [51], Montium [21], SGI Altix [3], HP ProLiant server [2], Convey hybrid-core HC-1 [4], RAMP [18], South Australia [58], BORPH [42], SHUM-uCOS [60];
I.2. Compiler techniques for ρMT: MT-ADRES [59], UltraSONIC [20];
II. *Implicit architectural ρMT support:*
CarCore Processor extensions [48], REDEFINE [38], Hthreads [35], Reconfigurable Multithreaded Architecture Model [55], University of Karlsruhe [10];
III. *Explicit architectural ρMT support*:
III.1. μ-architecture + OS: Reconfigurable Architectures of this kind are just emerging. This approach is promising for high performance efficient scheduling and execution of threads on reconfigurable hardware due to the hardware & software co-design of the ρMT managing mechanisms. OS4RS [29];
III.2. μ-architecture + compiler: Reconfigurable Multithreaded processor [27].

3 Design Problems & Open Research Questions

The very basic design questions related to thread scheduling on reconfigurable resources are:

- Which threads to execute, schedule or preempt at certain instance of time (e.g., when the requested reconfigurable area of prepared for execution hardware threads is higher than the available area)?
- Where to place a thread (in case of several possibilities)?
- When to reallocate the newly created threads and how to efficiently hide the reconfiguration latencies?

Depending on model assumptions, from complexity point of view, the scheduling problem on reconfigurable logic could be reduced to several well-known NP-Hard problems [44], [62], [9], [36]. Therefore, one of the ways to be solved is by reducing it to the well-known Bin-Packing problem, i.e., the scheduling problem could be solved by the introduction of an advanced heuristic algorithm. Some open research questions, partially and completely solved design problems, grouped by topic, are presented in the following subsections.

3.1 Hiding reconfiguration latencies

In reconfigurable systems, the reconfiguration latency is caused by the time needed for the configuration bitstream to set the reconfigurable device for the particular operation. Typically, configuration latency is introduced during the initial task loading (tasks are composed of one or multiple threads). This is one of the major system delays and causes severe performance degradation in case of frequent reconfigurations. In literature, the most common ways to hide or minimize the reconfiguration latency are:
1. Compressing the task's bitstream. Different techniques are examined in [37];
2. Employing prefetch technique and local caching for earlier reconfiguration (overlap reconfiguration with computations). The existing prefetch technique proposals are grouped into three categories:
– **Static** – predictions are performed at design time by the compiler (e.g., The Molen compiler [34]);
– **Dynamic** – at runtime by the reconfigurable scheduler, which stores most recent configurations [24];
– **Hybrid** (combining the Static & Dynamic approaches) [13], [24]. In case of misspredictions, alternative Hybrid methods [24] always pay time penalty, by delaying the reconfiguration. In [17] and [37], the authors propose inter-task placement in case of free reconfigurable area, but it is only limited to periodic hardware tasks. For aperiodic tasks, the problem has not been solved.

3.2 Optimized inter-thread communication scheme

The Erlangen-Nuremberg Slot Machine (ESM) [25] has target several problems common for contemporary FPGA based architectures such as: limitations of partial support on Actual FPGAs; I/O pin, intermodule communication and local memory dilemmas. The authors underline as a major advantage of the ESM platform its unique slot-based architecture which allows the slots to be used independently of each other by delivering peripheral data through a separate crossbar switch. The decision to exploit an off-chip crossbar is in order to have as many available resources on the FPGA for partially reconfigurable modules as possible. The ESM architecture is based on the flexible decoupling of the FPGA I/O-pins from a direct connection to an interface chip. This flexibility allows the independent placement of application modules in any available slot at run-time. As a result, run-time placement is not constrained by physical I/O-pin locations as the the I/O-pin routing.

3.3 Scheduling and placement algorithms

In the research work presented in [46] by ETH Zurich, Switzerland, the authors propose several algorithms to manage the sharing of resources in the reconfigurable surface. Their proposal includes system services for a partial reconfiguration, which by scheduling the dynamically incoming threads solve the problems with complex allocation situations. Detailed description of the

system model could be found in [45] and [53]. The primary idea of the project is to separate threads into two groups according to their arriving times - synchronous and arbitrary. For threads with aperiodic arriving times, the authors propose two non-preemptive techniques: "horizon" and "stuffing" methods [44]. On the other side, for threads with periodic arriving times, authors propose another two preemptive scheduling algorithms: "EDF-NF" and "MSDL" [15]. Unfortunately, the preemptive methods are not adaptable for threads with arbitrary arriving times, because the system cannot guarantee that each preempted thread, previously executed for some period of time, will finish before it's relative deadline. Each one of the scheduling techniques is combined with optimized placement method named "On the Fly Partitioning" [54], based on Bazargan partitioner [11].
The algorithms are further enhanced by a research group at Fundan University [62]. They introduce an advanced heuristic algorithms based on "stuffing" technique [44]. The authors prove that the combination of a scheduling algorithm with a recognition-complete placement method does not result to a recognition-complete technique. Therefore, they enhanced the "stuffing" scheduling algorithm [44] and named the new one: "windows-based stuffing". In [63], the authors propose "Compact Reservation" (CR) scheduling algorithm which attains recognition-earliest scheduling (arrange the start time of a newly arrived thread as early as possible) by exploiting the knowledge about temporal properties of each thread. In [6] the cases of potential thread migration depending on the workload is examined – a newly arrived thread is started either in software or in hardware. Slightly different approach is proposed in [17] by a research group at the Paderborn University. They enhance a single processor algorithm (e.g., a stochastic server) with preemption support (limited only during the time of reconfiguration) for hardware tasks.

3.4 Context switching

In [22], the authors clearly identify the two possible techniques for context switching of hardware threads in partially reconfigurable FPGAs. The techniques are named as follows:
1) *Thread Specific Access Structures* – when the scheduler decides to switch a thread, it's current state is saved in an external structure. The major advantages of this approach are the high data efficiency and its architecture independence. The disadvantages come from the fact that each thread is different and it is difficult to design a standard generic interface. On the other hand, the designer also needs detailed knowledge about the structure and the behavior of the thread, therefore the method is not applicable for IP Cores represented by black-box functional blocks. In [41], the authors explore the control software required to support thread switching as well as the requirements and features of context saving and restoring in the FPGA coprocessor context. Similar approach is taken in [5] - each hardware thread is represented by one complicated Finite State Machine (FSM). In case of context switching, the scheduler saves current FSM state together with multiple data registers.

2) *Configuration Port Access* – the thread bitstream is completely downloaded from the FPGA chip and the state information is filtered. In [22], the authors design custom tools for offline bitstream processing. The advantages of the approach are: additional design efforts and information about internal thread behavior are not needed. In [5], the authors additionally compress the bitstream (bitwise XOR) to minimize the size and delay of downloaded data. The method is named "ReadBack technique".

3.5 Real-time support for reconfigurable hardware threads

In the literature, there are two basic approaches (described below) capable to deliver real-time support for software/ hardware heterogeneous platforms:
1) *Per-case solutions using Heuristic Algorithms* – many of the proposed algorithms (see Subsection 3.3) support "Commitment Test" - each newly created hardware thread is checked for successful termination before its deadline and critical affects (e.g., delays) on other executing threads. Unfortunately the proposed ideas (heuristic algorithms) are designed only for independent hardware threads with known executing times, therefore they are not applicable for hardware threads with data, resource or communication dependencies.
2) *Complete Solutions on Conventional Reconfigurable Platforms* (e.g., BORPH [42], UltraSonic [20], Hthreads [35]) – none of them supports reconfigurable resource sharing among executing threads. In case reconfigurable area is shared, all possible resource collisions are solved at compile time.

3.6 Run-time creation and termination of threads

Currently, all existing proposals (Section 3) offer partial solution for scheduling non-preemptive and periodic preemptive only tasks on reconfigurable logic. Therefore, the open question arises: "How to manage creation and termination of data, resource and communication dependent real-time threads?". The topic is still open and it is closely related to Subsection 3.5. It is one of our primary objectives to make further investigations in future research. Many of the current projects (e.g., [25], [42]) have run-time creation as a feature, but none of them provides resource sharing and real-time support for data dependent threads.

3.7 Hardware scheduler agnostic to the employed embedded GPP

The main research question still stays open: "When and under what circumstances the hardware scheduler should initialize a communication to the GPP?". All current solutions are for specific architectures and do not provide general holistic approach to the scheduling problem.
Table 1 summarizes the design problems and open research question discussed in this section.

Table 1: Design Problems & Open Research Questions

Partially [PS] & Completely [CS] Solved Design Problems:
[CS] - Hiding reconfiguration latencies by prefetching, context switching and resource reusage among threads; [13], [24], [37]
[PS] - Optimized inter-thread communication scheme; [25]
[PS] - Real-time thread support by the reconfigurable architecture; [48], [42], [20], [35]
[PS] - Preemptive techniques [context switching] for threads with arbitrary arriving times. Consider inter-thread data dependencies, free reconfigurable area and communication profile; [46], [45], [62], [44]
[PS] - Thread migration between software and hardware; [41], [22], [5]
[PS] - Consider virtualization and protection; [56], [55]
[PS] - Rescheduling of threads, depending on the workload; [17]
[PS] - Run-time creation and termination of threads; [25], [42]
Open Research Questions [O]:
[O] - Hardware scheduler agnostic to the employed embedded GPP processor;
[O] - System performance evaluation parameters;
[O] - Intra-thread management by the scheduler;

3.8 Application Prospective & Potential Research Directions

One of the direct gains from employing a ρMT architecture, after solving the open questions from Section 3, would be the capability for time efficient run-time creation, termination and management of multiple threads sharing the reconfigurable resources without critically affecting (delaying) each other. Possible future research could extend the functionality and overcome some limitations providing, e.g.:

1. Real-time and runtime support of multiple hardware threads through architecture agnostic hardware scheduler. It could support run-time creation and termination of multiple threads mapped into reconfigurable logic and hardware system implementation. The compiler would be only responsible for inter-thread optimizations. The hardware scheduler would manage intra-thread optimizations;
2. More sophisticated scheduling policies capable to fairly distribute resources among multiple resource-dependent hardware threads. Introduction of a metric evaluating the resource distribution and potential thread starvation.
3. Hiding of reconfiguration latencies and efficient thread-preemption and migration model with estimation of performance costs. For periodic and sporadic threads, the migration might take place right after the end of the current iteration.

4 Conclusions

In this report, we provided a survey and proposed a taxonomy of existing reconfigurable architectures with respect to their support of multithreading on reconfigurable resources. We identified three main classes – *explicit*, *implicit* and *no* ρMT *support*, each one of them with several sub-categories. We further summarized a number of identified design problems and several research questions, which addressed performance efficient management, mapping, sharing, scheduling and execution of threads on reconfigurable hardware resources. We provided our vision for potential research directions and possible solutions of some open research topics. We marked which of the identified design problems have been partially or completely solved and which research questions remain open.

Acknowledgments

This work was supported by the HiPEAC European Network of Excellence - cluster 1200 (FP6-Contract number IST-004408) and by the Dutch Technology Foundation STW, applied science division of NWO (project DSC.7533).

References

[1] http://ramp.eecs.berkeley.edu/.

[2] http://www.hp.com/products/servers/platforms/.

[3] http://www.sgi.com/products/servers/altix/.

[4] The convey HC-1 computer, architecture overview (white paper)- http://www.conveycomputer.com, 2008.

[5] A. Ahmadinia, C. Bobda, D. Koch, M. Majer, and J. Teich. Task scheduling for heterogeneous reconfigurable computers. In *SBCCI*, pages 22–27, 2004.

[6] A. Ahmadinia, C. Bobda, and J. Teich. Online placement for dynamically reconfigurable devices. *IJES*, 1(3/4):165–178, 2005.

[7] M. Alle, K. Varadarajan, R. C. Ramesh, J. Nimmy, A. Fell, A. Rao, S. K. Nandy, and R. Narayan. Synthesis of application accelerators on runtime reconfigurable hardware. In *ASAP*, pages 13–18, 2008.

[8] D. Andrews, D. Niehaus, R. Jidin, M. Finley, W. Peck, M. Frisbie, J. Ortiz, E. Komp, and P. Ashenden. Programming models for hybrid FPGA-CPU computational components: a missing link. *IEEE Micro*, 24:42–53, 2004.

[9] J. Angermeier and J. Teich. Heuristics for Scheduling Reconfigurable Devices with Consideration of Reconfiguration Overheads. In *Proceedings 15th Reconfigurable Architectures Workshop*, Miami, Florida, 2008.

[10] L. Bauer, M. Shafique, S. Kreutz, and J. Henkel. Run-time system for an extensible embedded processor with dynamic instruction set. In *DATE*, pages 752–757, 2008.

[11] K. Bazargan, R. Kastner, and M. Sarrafzadeh. Fast template placement for reconfigurable computing systems. *IEEE Design & Test of Computers*, 17(1):68–83, 2000.

[12] R. Chaves. *Secure Computing on Reconfigurable Systems*. PhD thesis, TU Delft, December 2007.

[13] Y. Chen and S. Y. Chen. Cost-driven hybrid configuration prefetching for partial reconfigurable coprocessor. In *IPDPS*, pages 1–8. IEEE Press, 2007.

[14] K. Compton and S. Hauck. Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, 2002.

[15] K. Danne and M. Platzner. A heuristic approach to schedule periodic real-time tasks on reconfigurable hardware. In *FPL*, pages 568–573. IEEE Press, 2005.

[16] O. Diessel and G. B. Wigley. Opportunities for operating systems research in reconfigurable computing. In *ACRC*, 1999.

[17] F. Dittmann. *Methods to Exploit Reconfigurable Fabrics - Making Reconfigurable Systems Mature*. PhD thesis, University of Paderborn, 2007.

[18] G. Gibeling, A. Schultz, and K. Asanovic. The RAMP architecture & description language. In *WARFP*, 2006.

[19] J. Glossner, M. Schulte, M. Moudgill, D. Iancu, S. Jinturkar, T. Raja, G. Nacer, and S. Vassiliadis. Sandblaster low-power multithreaded sdr baseband processor. In *WASP*, pages 53–58, 2004.

[20] S. D. Haynes, H. G. Epsom, R. J. Cooper, and P. L. McAlpine. UltraSONIC: A reconfigurable architecture for video image processing. In *FPL'02*, pages 482–491. Springer-Verlag, 2002.

[21] P. M. Heysters. Coarse-grained reconfigurable computing for power aware applications. In *ERSA*, pages 272–280, 2006.

[22] H. Kalte and M. Porrmann. Context saving and restoring for multitasking in reconfigurable systems. In *FPL*, pages 223–228. IEEE Press, 2005.

[23] G. K. Kuzmanov, S. Vassiliadis, and J. T. J. van Eijndhoven. Hardwired MPEG-4 repetitive padding. *IEEE Transactions on Multimedia*, 7:261–268, April 2005.

[24] Z. Li and S. Hauck. Configuration prefetching techniques for partial reconfigurable coprocessor with relocation and defragmentation. In *FPGA*, pages 187–195, 2002.

[25] M. Majer, J. Teich, A. Ahmadinia, and C. Bobda. The Erlangen Slot Machine: A dynamically reconfigurable fpga-based computer. *VLSI Signal Processing*, 47(1):15–31, 2007.

[26] S. Mamadi. *Reconfigurable Multi Processors for Programmable Communication Systems*. PhD thesis, Wisconsin-Madison, 2006.

[27] S. Mamidi, M. Schulte, D. Iancu, and J. Glossner. Architecture support for reconfigurable multithreaded processors in programmable communication systems. In *ASAP*, pages 320–327. IEEE Press, 2007.

[28] T. Marescaux, J.-Y. Mignolet, A. Bartic, W. Moffat, D. Verkest, S. Vernalde, and R. Lauwereins. Networks-on-chip as hardware components of an OS for reconfigurable systems. In *FPL*, volume 2778 of *LNCS*, pages 595–605. Springer, 2003.

[29] T. Marescaux, V. Nollet, J.-Y. Mignolet, A. Bartic, W. Moffat, P. Avasare, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Run-time support for heterogeneous multitasking on reconfigurable SoCs. *Integration*, 38(1):107–130, 2004.

[30] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins. DRESC: a retargetable compiler for coarse-grained reconfigurable architectures. In *FPT*, pages 166–173, 2002.

[31] J. Noguera and R. M. Badia. Multitasking on reconfigurable architectures: microarchitecture support and dynamic scheduling. *Trans. on Embedded Computing Sys.*, 3(2):385–406, 2004.

[32] V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Designing an Operating System for a heterogeneous reconfigurable SoC. In *IPDPS*, pages 174–180, 2003.

[33] V. Nollet, J.-Y. Mignolet, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins. Hierarchical run-time reconfiguration managed by an Operating System for reconfigurable systems. In *Engineering of Reconfigurable Systems and Algorithms*, pages 81–87. CSREA Press, 2003.

[34] E. M. Panainte. *The Molen Compiler for Reconfigurable Architectures*. PhD thesis, TU Delft, 2007.

[35] W. Peck, E. Anderson, J. Agron, J. Stevens, F. Baijot, and D. Andrews. HTHREADS: a computational model for reconfigurable devices. In *FPL*, pages 885–888, 2006.

[36] J. Resano, D. Mozos, and F. Catthoor. A hybrid prefetch scheduling heuristic to minimize at run-time the reconfiguration overhead of dynamically reconfigurable hardware. In *DATE*, pages –, 2005.

[37] J. Resano, D. Mozos, D. Verkest, and F. Catthoor. A reconfiguration manager for dynamically reconfigurable hardware. *IEEE Design & Test of Computers*, 22(5):452–460, 2005.

[38] A. Satrawala, K. Varadarajan, M. Lie, S. Nandy, and R. Narayan. Redefine: Architecture of a soc fabric for runtime composition of computation structures. In *FPL*, pages 558–561, 2007.

[39] K. Seno and M. Yamazaki. Virtual mobile engine (VME) LSI that "changes its spots" achievies ultralow power and diverse functionality. *CX-News - http://www.sony.com*, 42, 2005.

[40] M. Sima, S. Vassiliadis, S. D. Cotofana, J. T. J. van Eijndhoven, and K. A. Vissers. Field-programmable custom computing machines - a taxonomy. In *FPL'02*, pages 79–88, 2002.

[41] H. Simmler and L. Levinson. Multitasking on FPGA coprocessors. In *FPL*, pages 121–130. Springer-Verlag, Nov. 2000.

[42] H. K.-H. So and R. Brodersen. A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH. *ACM Transactions on Embedded Computing Systems*, 7(2):1401–1407, 2008.

[43] H. K.-H. So and R. W. Brodersen. *BORPH: An Operating System for FPGA-Based Reconfigurable Computers*. PhD thesis, EECS Department, University of California, Berkeley, 2007.

[44] C. Steiger, H. Walder, and M. Platzner. Heuristics for online scheduling real-time tasks to partially reconfigurable devices. In *FPL*, pages 575–584, 2003.

[45] C. Steiger, H. Walder, and M. Platzner. Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks. *IEEE Trans. Computers*, 53(11):1393–1407, 2004.

[46] C. Steiger, H. Walder, M. Platzner, and L. Thiele. Online scheduling and placement of real-time tasks to partially reconfigurable devices. In *RTSS*, pages 224–235. IEEE Computer Society, 2003.

[47] Z. Tan. Multithreaded sparc v8 functional model for ramp gold. In *Presentation - http://ramp.eecs.berkeley.edu/*, page 15, 2008.

[48] S. Uhrig, S. Maier, G. K. Kuzmanov, and T. Ungerer. Coupling of a reconfigurable architecture and a multithreaded processor core with integrated real-time scheduling. In *RAW*, pages 209–217, 2006.

[49] S. Uhrig, S. Maier, and T. Ungerer. Toward a processor core for real-time capable autonomic systems. In *ISSPIT-05*, pages 19–22, 2005.

[50] T. Ungerer, B. Robic, and J. Silc. A survey of processors with explicitcit multithreading. *ACM Computing Surveys*, 35(1):29–63, 2003.

[51] S. Vassiliadis, S. Wong, and S. D. Cotofana. The MOLEN $\mu\rho$-coded processor. In *(FPL), Springer-Verlag (LNCS) Vol. 2147*, pages 275–285, August 2001.

[52] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. K. Kuzmanov, and E. M. Panainte. The Molen polymorphic processor. *IEEE Transactions on Computers*, 53:1363–1375, November 2004.

[53] H. Walder and M. Platzner. Reconfigurable hardware Operating Systems: From design concepts to realizations. In *Engineering of Reconfigurable Systems and Algorithms*, pages 284–287. CSREA Press, 2003.

[54] H. Walder, C. Steiger, and M. Platzner. Fast online task placement on FPGAs: Free space partitioning and 2D-Hashing. In *IPDPS'03*, pages 178–178, 2003.

[55] S. Wallner. A reconfigurable multi-threaded architecture model. In *APC-SAC*, volume 2823, pages 193–207. Springer, 2003.

[56] S. Wallner. Micro-task processing in heterogeneous reconfigurable systems. *J. Comput. Sci. Technol*, 20(5):624–634, 2005.

[57] T. Wiangtong, P. Y. K. Cheung, and W. Luk. Cluster-driven hardware/software partitioning and scheduling approach for a reconfigurable computer system. In *FPL*, pages 1071–1074, 2003.

[58] G. B. Wigley and D. A. Kearney. The first real operating system for reconfigurable computers. In *ACSAC*, pages 129–136. IEEE Computer Society Press, Jan. 2000.

[59] K. Wu, A. Kanstein, J. Madsen, and M. Berekovic. MT-ADRES: Multithreading on coarse-grained reconfigurable architecture. In *ARC*, volume 4419 of *LNCS*, pages 26–38. Springer, 2007.

[60] B. Zhou, W. Qui, and C.-L. Peng. An operating system framework for reconfigurable systems. In *CIT*, pages 781–787, 2005.

[61] B. Zhou, W. Qui, and C.-L. Peng. Shum-ucos: A rtos using multi-task model to reduce migration cost between sw/hw tasks. In *CSCWD*, pages 984–989, 2005.

[62] X. Zhou, Y. Wang, X.-Z. Huang, and C.-L. Peng. On-line scheduling of real-time tasks for reconfigurable computing system. In *FPT*, pages 57–64, 2006.

[63] X. Zhou, Y. Wang, X.-Z. Huang, and C.-L. Peng. Fast on-line task placement and scheduling on reconfigurable devices. In *FPL*, pages 132–138. IEEE Computer Society, 2007.