Reconfigurable Multithreading Architectures: A Survey

Pavel G. Zaykov, Georgi K. Kuzmanov, Georgi N. Gaydadjiev

Computer Engineering, EEMCS, TU Delft, The Netherlands
{ P. Zaykov, G.K.Kuzmanov, G.N.Gaydadjiev} ∋ tudelft·nl

Abstract. This paper provides a survey on the existing proposals in the field of reconfigurable multithreading (ρMT) architectures. 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. 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

Many applications running on modern embedded devices are composed of multiple threads, typically processing (exchanging) data among multiple sources. 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 reconfigurable technology [1], [2], [3]. 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 [4], 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 paper we approach the reconfigurable multithreading (ρMT) architectural

problems both from the hardware and the software prospective. The specific contributions of this paper 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*;
– 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 paper 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 [5]. 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 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 what is used in GPP systems. In general purpose systems, the classification is based on multithreading support from algorithmic point of view [4]. 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. [1], [6], [7];
– Reconfigurable Interleaved Multithreading (ρIMT): e.g. [8];
– Reconfigurable Simultaneous Multithreading (ρSMT): e.g. [9], [10];
In this paper, 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 at the

architectural and 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 concep-

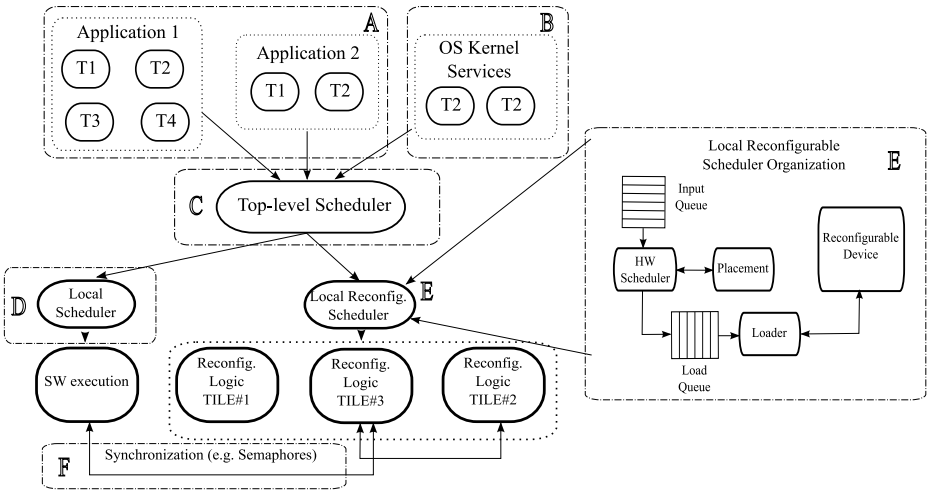

Fig. 1. A Conceptual Behavioral Model of ρMT Related Projects

tual behavioral model of an ρMT system is depictured in Figure 1. It 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 [11], 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 [11], 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 [1] 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. In the Molen original papers, multithreading has not been discussed, but a follow-up research towards multithreading has been reported in [9]. 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 processor core, we choose Montium TP [2], designed by RECORE Systems. This architecture has the following features: 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 processor was originally targeting the domain of streaming applications.

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.

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 [6] and [12] by a research group at the University of South Australia.

BORPH [13]: The research work presented 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 proposal has the following limitations - hardware executed 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 [14]: Another design, tackling the problems caused by the essential differences between software and hardware-tasks is the SHUM-uCOS by the Fudan University, China [14]. The authors propose a real-time OS (RTOS) for reconfigurable systems employing an 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. The limitations on the current implementation of SHUM-uCOS are in the static scheduling approach and the resource reusage, supported by the compiler only. For evaluation of the system, the authors use benchmarks and VOIP algorithms.
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 [7]: In MT-ADRES by IMEC, Belgium, the compiler framework 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. All experiments providing information about MT-ADRES performance are achieved through multimedia simulations.
UltraSONIC [15]: Another proposal falling in this category is the UltraSONIC project, represented by Sony Research Labs, UK. It is a reconfigurable architecture optimized for video processing. It has a list of Plug-In Processing Elements, connected through several buses. The programmer receives an architecture abstraction through an API interface. Multitasking is achieved through an algorithm (implemented in the compiler) working on the application Directed Acyclic Graph.

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 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.
Reconfigurable Extensions for the CarCore Processor: In [9], the authors combine a simultaneous multithreading (SMT) processor with a Molen style reconfigurable coprocessor [1]. 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. 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 active threads.

Hthreads [10]: The Hthreads(Hybrid Threads) model presented by University of Kansas 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 migration of thread management, synchronization primitives and run-time scheduling services (Figure 1, Section F) for both hardware and software threads into dedicated hardware module accessed from the GPP only through an universal bus. The authors represent hardware threads with user defined component, state controller and universal interface (Register Set). Synchronization procedures are performed through semaphores. 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.

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 [16]: In [16], 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 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 (OS4RS). The system maintains several threads by a two-level scheduler. In their current implementation, the top-level scheduler (Figure 1, Section C) is implemented in software. The low-level/ local scheduler can be implemented in software (Figure 1, Section D) or hardware (Figure 1, Section E) depending on the type of the slave computing resources (GPP or reconfigurable logic). In their current implementation, the local-level hardware (reconfigurable) scheduler 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 OS4RS has been tested in JPEG frame decoding and experimental 3D video game.

Reconfigurable Multithreaded processor [8] by University of Wisconsin-Madison. The authors augment multiprocessor system, composed by multithreaded Digital Signal Processor(DSP) and RISC processor, with multiple Polymorphic Hardware Accelerators (PHAs) - reconfigurable hardware units. 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 procedure. The multithreading is mainly employed to hide the reconfiguration time. 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. The architecture is limited to Interleaved 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 propose two PHA binding techniques - static & dynamic. The implementation includes only static (compile time) mapping approach. In case of a run-time binding, the system should provide 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. More elaborated discussion and full list of references could be found in [17]:
I. *No architectural ρMT support:*
I.1. OS support for ρMT: Molen [1], Montium [2], Convey hybrid-core HC-1 [18], RAMP [19], South Australia [6], BORPH [13], SHUM-uCOS [14];
I.2. Compiler techniques for ρMT: MT-ADRES [7], UltraSONIC [20];
II. *Implicit architectural ρMT support*:
CarCore Processor extensions [9], REDEFINE [21], Hthreads [10], ρMT Architectural Model [22], University of Karlsruhe [23];
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 [16];
III.2. μ-architecture + compiler: Reconfigurable Multithreaded processor [8].

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 [24], [25], [26]. Therefore, one of the ways to be solved is by introduction of an advanced heuristic algorithm. Some open research questions and several partially and completely solved design problems, grouped by topic, are presented below. For more details, the interested reader is referred to [17].

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 [27];
2. Employing prefetch technique for earlier reconfiguration (overlap with computation) and local caching. The existing proposals could be grouped into three categories:

– **Static** – predictions are performed at design time by the compiler; (e.g., The Molen compiler [28]);

– **Dynamic** – at runtime by the reconfigurable scheduler, which stores most recent configurations [29];

– **Hybrid** (combining the Static & Dynamic approaches) [29]. In case of missprediction, alternative Hybrid methods [29] always pay time penalty, by delaying the reconfiguration;

Scheduling and placement algorithms: In the research work presented in [30] by ETH Zurich, 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. The primary idea of the project is to separate threads into two groups according to their arriving times - synchronous and arbitrary, which are scheduled by different heuristic algorithms. Each one of the scheduling techniques is combined with optimized placement methods.

The algorithms are further enhanced by a research group at Fundan University [25]. The authors prove that the combination of a scheduling algorithm with a recognition-complete placement method does not result to a recognition-complete technique. They also investigate the cases of potential thread migration – a newly arrived thread is started either in software or in hardware. Slightly different approach is proposed in [31] 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.

Context switching: In [32], the authors clearly identify the two possible tech-

niques 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 the approach are the high data efficiency and the architecture independence. The disadvantages come from the fact that each thread is different and it is difficult to design a standard generic interface. In [33], 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 [34] - each hardware thread is represented by one complicated finite state machine.
2) *Configuration Port Access* – the thread bitstream is completely downloaded from the FPGA chip and the state information is filtered. In [32], the authors design custom tools for offline bitstream processing. The advantages of this approach is that additional design efforts and its information about internal thread behavior are not needed. In [35], the authors additionally compress the bitstream to minimize the size and delay of downloaded data.
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 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 [13], UltraSonic [20], Hthreads [10]) – 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.
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, 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.
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 mi-

gration model with estimation of performance costs. For periodic and sporadic threads, the migration might take place right after the end of the current iteration. The following list summarize the topics presented in Section 3:
Partially [PS] & Completely[CS] Solved Design Problems:
[CS] - Hiding reconfiguration latencies by prefetching, context switching and resource reusage among threads; [36], [29], [27]
[PS] - Optimized inter-thread communication scheme; [34]
[PS] - Real-time thread support by the reconfigurable architecture; [9], [20], [10]
[PS] - Preemptive techniques [context switching] for threads with arbitrary arriving times. Consider inter-thread data dependencies, free reconfigurable area and communication profile; [30], [25], [24]
[PS] - Thread migration between software and hardware; [33], [32], [35]
[PS] - Consider virtualization and protection; [37], [22]
[PS] - Rescheduling of threads, depending on the workload; [31]
[PS] - Run-time creation and termination of threads; [34], [13]
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;

4 Conclusions

In this paper, 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. S. Vassiliadis, S. Wong, and S. D. Cotofana, "The MOLEN $\mu\rho$-coded processor," in *(FPL), Springer-Verlag (LNCS) Vol. 2147*, August 2001, pp. 275–285.
2. P. M. Heysters, "Coarse-grained reconfigurable computing for power aware applications," in *ERSA*, 2006, pp. 272–280.

3. 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*, vol. 42, 2005.
4. T. Ungerer, B. Robic, and J. Silc, "A survey of processors with expliclicit multithreading," *ACM Computing Surveys*, vol. 35(1), pp. 29–63, 2003.
5. 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*, 2002, pp. 79–88.
6. G. B. Wigley and D. A. Kearney, "The first real operating system for reconfigurable computers," in *ACSAC*. IEEE Computer Society Press, Jan. 2000, pp. 129–136.
7. K. Wu, A. Kanstein, J. Madsen, and M. Berekovic, "MT-ADRES: Multithreading on coarse-grained reconfigurable architecture," in *ARC*, ser. LNCS, vol. 4419. Springer, 2007, pp. 26–38.
8. S. Mamidi, M. Schulte, D. Iancu, and J. Glossner, "Architecture support for reconfigurable multithreaded processors in programmable communication systems," in *ASAP*. IEEE Press, 2007, pp. 320–327.
9. 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*, 2006, pp. 209–217.
10. W. Peck, E. Anderson, J. Agron, J. Stevens, F. Baijot, and D. Andrews, "HTHREADS: a computational model for reconfigurable devices," in *FPL*, 2006, pp. 885–888.
11. K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, 2002.
12. O. Diessel and G. B. Wigley, "Opportunities for operating systems research in reconfigurable computing," in *ACRC*, 1999.
13. 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*, vol. 7, no. 2, pp. 1401–1407, 2008.
14. B. Zhou, W. Qui, and C.-L. Peng, "An operating system framework for reconfigurable systems," in *CIT*, 2005, pp. 781–787.
15. J. Noguera and R. M. Badia, "Multitasking on reconfigurable architectures: microarchitecture support and dynamic scheduling," *Trans. on Embedded Computing Sys.*, vol. 3, no. 2, pp. 385–406, 2004.
16. 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*, vol. 38(1), pp. 107–130, 2004.
17. P. G. Zaykov, G. K. Kuzmanov, and G. N. Gaydadjiev, "State-of-the-art reconfigurable multithreading architectures," Technical Report - CE-TR-2009-02, 2009.
18. "The convey HC-1 computer, architecture overview (white paper)- http://www.conveycomputer.com," p. 11, 2008.
19. G. Gibeling, A. Schultz, and K. Asanovic, "The RAMP architecture & description language," in *WARFP*, 2006.
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*. Springer-Verlag, 2002, pp. 482–491.
21. 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*, 2007, pp. 558–561.

22. S. Wallner, "A reconfigurable multi-threaded architecture model," in *APCSAC*, vol. 2823. Springer, 2003, pp. 193–207.
23. L. Bauer, M. Shafique, S. Kreutz, and J. Henkel, "Run-time system for an extensible embedded processor with dynamic instruction set," in *DATE*, 2008, pp. 752–757.
24. C. Steiger, H. Walder, and M. Platzner, "Heuristics for online scheduling real-time tasks to partially reconfigurable devices," in *FPL*, 2003, pp. 575–584.
25. X. Zhou, Y. Wang, X.-Z. Huang, and C.-L. Peng, "On-line scheduling of real-time tasks for reconfigurable computing system," in *FPT*, 2006, pp. 57–64.
26. J. Angermeier and J. Teich, "Heuristics for Scheduling Reconfigurable Devices with Consideration of Reconfiguration Overheads," in *Proceedings 15th Reconfigurable Architectures Workshop*, Miami, Florida, 2008.
27. J. Resano, D. Mozos, D. Verkest, and F. Catthoor, "A reconfiguration manager for dynamically reconfigurable hardware," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 452–460, 2005.
28. E. M. Panainte, "The Molen compiler for reconfigurable architectures," Ph.D. dissertation, TU Delft, 2007.
29. Z. Li and S. Hauck, "Configuration prefetching techniques for partial reconfigurable coprocessor with relocation and defragmentation," in *FPGA*, 2002, pp. 187–195.
30. C. Steiger, H. Walder, M. Platzner, and L. Thiele, "Online scheduling and placement of real-time tasks to partially reconfigurable devices," in *RTSS*. IEEE Computer Society, 2003, pp. 224–235.
31. F. Dittmann, "Methods to exploit reconfigurable fabrics - making reconfigurable systems mature," Ph.D. dissertation, University of Paderborn, 2007.
32. H. Kalte and M. Porrmann, "Context saving and restoring for multitasking in reconfigurable systems," in *FPL*. IEEE Press, 2005, pp. 223–228.
33. H. Simmler and L. Levinson, "Multitasking on FPGA coprocessors," in *FPL*. Springer-Verlag, Nov. 2000, pp. 121–130.
34. M. Majer, J. Teich, A. Ahmadinia, and C. Bobda, "The Erlangen Slot Machine: A dynamically reconfigurable fpga-based computer," *VLSI Signal Processing*, vol. 47, no. 1, pp. 15–31, 2007.
35. A. Ahmadinia, C. Bobda, D. Koch, M. Majer, and J. Teich, "Task scheduling for heterogeneous reconfigurable computers," in *SBCCI*, 2004, pp. 22–27.
36. Y. Chen and S. Y. Chen, "Cost-driven hybrid configuration prefetching for partial reconfigurable coprocessor," in *IPDPS*. IEEE Press, 2007, pp. 1–8.
37. S. Wallner, "Micro-task processing in heterogeneous reconfigurable systems," *J. Comput. Sci. Technol*, vol. 20, no. 5, pp. 624–634, 2005.