Run-Time Resource Allocation for Simultaneous Multi-Tasking in Multi-Core Reconfigurable Processors

Waheed Ahmed, Muhammad Shafique, Lars Bauer, Manuel Hammerich, Jörg Henkel, and Juergen Becker
Karlsruhe Institute of Technology, Karlsruhe, Germany
{ahmed.waheed, muhammad.shafique, lars.bauer, henkel, becker} @ kit.edu

State-of-the-art multi-core reconfigurable processors do not exploit the full potential of simultaneous multi-tasking with run-time adaptive reconfigurable fabric allocation. We propose a novel run-time system for simultaneous multi-tasking in a multi-core reconfigurable processor that adaptively allocates the mixed-grained reconfigurable fabric resource at run time among different tasks considering their performance constraints. Our scheme employs the novel concept of refined task-criticality (based on the functional-block-level performance constraints) considering the computational properties of dependent tasks and their inherent potential for acceleration. Our scheme dynamically compensates the deadline misses at the functional block level. It thereby reduces the potential task-level deadline misses under competing scenarios. With the help of a secure video conferencing application (with 4 dependent tasks of diverse computational properties), we demonstrate that our scheme reduces the deadline misses by (on average) 6x under given performance constraints, when compared to state-of-the-art reconfigurable processors [1, 9, 12].

I. Introduction and Motivation

Modern multi-core reconfigurable processors are mixed-grained ([1], 4S [8], Morpheus [9]), i.e., they are composed of various (RISC-like) cores coupled with multiple fine-grained and coarse-grained reconfigurable fabrics in order to expedite bit-/byte-level operations (bit shuffling, packing, merging, etc.) and (sub)-word-level operations (add, subtract, multiply, etc.), respectively. There is a need to efficiently support *simultaneous multi-tasking* with run-time mixed-grained fabric allocation in multi-core reconfigurable processors for exploiting the *Task-Level Parallelism*. Multi-core reconfigurable processors may exploit:

a) task-level parallelism by executing multiple tasks on different cores in parallel, and

b) instruction-level parallelism by accelerating the functional blocks of applications (that may contain diverse computational kernels[1]) using Instruction Set Extensions (ISEs) to the cores [1,8,9,12,15]. These ISEs consist of data-paths that may be reconfigured on parts of a fine-grained or a coarse-grained reconfigurable fabric, which are shared among different cores. It thereby enables each kernel to have different ISEs with different performance vs. area tradeoffs, i.e., different speedups for different utilizations of the fabric.

However, state-of-the-art multi-core reconfigurable processors [1,8,9,12] do not exploit simultaneous multi-tasking (with both task- and instruction-level parallelism) at their full throttle, when considering the heterogeneous nature of fine- and coarse-grained fabrics and the scenarios of run-time varying (i) task mapping, (ii) workload conditions, etc. Moreover, these processors lack efficient (in terms of performance, for instance) fabric allocation schemes that operate with consideration of simultaneous multi-tasking. We are the first to explore the full potential of simultaneous multi-tasking in multi-core reconfigurable processors by allocating the mixed-grained reconfigurable fabrics among different tasks at the granularity level of functional blocks.

When exploiting the task-level parallelism for simultaneous multi-tasking, the case of dependent tasks needs to be distinguished. Dependent tasks share a common deadline and priority (associated with the application). For example, in video conferencing application, the video and audio encoding and decoding tasks are considered as dependent tasks as they share a common deadline (30 frames/sec) and they depend on each other as they cannot start their next job until both of them do not complete their job. Dependent tasks significantly differ in their memory requirements and computational properties. Some tasks exhibit control-dominant and/or bit/byte-level processing (bit shuffling, packing, merging, etc.), while some other tasks exhibit data-dominant and/or (sub) word-level processing. Therefore, different tasks might require less or more area of fine-grained and/or coarse-grained fabrics (to expedite their functional blocks) depending upon their inherent potential for acceleration. Furthermore, the requirements of these tasks may vary frequently at run time, e.g., due to a change in the input data, which may lead to the situation where one or more tasks may become critical in determining the overall Quality of Service (QoS) of dependent tasks. The task is termed as *critical* if it misses or going to miss the deadline and the margin with which the task misses (or going to miss) the deadline denotes the *task's crticality*. Recently, the notion of criticality has been evolved for the dependent tasks to perform load balancing and frequency scaling [3]. However, criticality has not been fully explored by run-time schemes for reconfigurable fabric allocation, especially in case of simultaneous multi-tasking.

Hence, it can be concluded that to enable simultaneous multi-tasking on multi-core reconfigurable processors under high throughput and QoS constraints (that may vary at run time), a run-time system is desirable that efficiently allocates the mixed-grained fabric resource (at run-time) among different competing dependent tasks, while considering their respective criticalities to ensure that each task achieves its performance constraint (finishing its job before deadline), or alternatively missing the deadline by a user-defined tolera-

[1] The compute-intensive loops, which are executed most often

978-0-7695-4301-7/11 $26.00 © 2011 IEEE
DOI 10.1109/FCCM.2011.46

ble margin (i.e., graceful degradation). We therefore consider a system with soft constraints.

Our Novel Contribution: We propose a novel *Processor Control Unit* as an integral run-time system for multi-core reconfigurable processors. It ensures the application's QoS by dynamic deadline adaptations and mixed-grained fabric allocation to simultaneous multiple tasks at the functional block level. The proposed processor control unit achieves this using its two major components:

a) Constraints-based Fabric-Allocator: It dynamically allocates the mixed-grained fabric to the functional blocks based on their criticalities, priorities, deadlines and computational properties.

b) QoS Controller: It monitors and updates the state of the system and estimates the criticality of critical tasks and fine-tunes the deadlines at functional block level.

II. Related Work

The authors in [5,10] have discussed the operating system based task's placement and scheduling of complete tasks on the reconfigurable fabric, while considering the reconfiguration latency in order to achieve higher performance at run-time. In contrast, approaches discussed in [6,7] targeted reconfigurable architectures for high performance computing by enabling the sharing of attached FPGAs among parallel applications at compile-time. These approaches are beneficial in case that the processing behavior of tasks is not changed during their run-time. Authors in [11] presented an approach that maps different kernels of a functional block onto a coarse-grained reconfigurable fabric while mapping decisions are determined at compile-time. The above-discussed approaches either lack run-time adaptivity at functional block and/or allocate a certain type of fabric to the complete task, thus limiting the exploitation of the full potential of simultaneous multi-tasking in multi-core reconfigurable processors. Altogether, none of the presented approaches provides a run-time system to efficiently enable simultaneous multi-tasking with mixed-grained reconfigurable fabric allocation at functional block level, when considering dependent tasks and run-time varying scenarios.

III. Our Multi-core Reconfigurable Processor

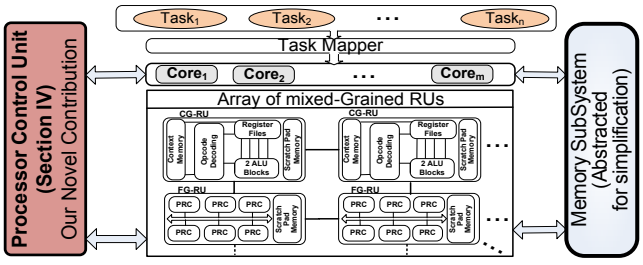

Fig. 1 Model of our multi-core reconfigurable processor [1]

Fig. 1 shows the integratation of our novel *Processor Control Unit* with the multi-core reconfigurable processor, which consists of several RISC cores tightly-coupled with mixed-grained fabrics, i.e., an array of fine-grained (FG)- and coarse-grained (CG)- reconfigurable units (RUs). A FG-RU is realized as an embedded FPGA, which consists of *Partially Reconfigurable Containers* (PRCs) that can be reconfigured to contain different hardware accelerators. Both FG- and CG-RUs have dedicated scratch pad memories – connected to the Memory Sub-System – to allow for fast data access and to store intermediate results. The Task Mapper maps an incoming task to a RISC core. Different RISC cores can execute their tasks in parrallel and use RUs to expedite them.

IV. Processor Control Unit

The Processor Control Unit (as an integral run-time system in our multi-core reconfigurable processor) reacts to different scenarios, while considering the system constraints, the current load on the system, and the utilized resources. It therefore enables the possibility to bring the system to perform with required QoS under run-time varying scenarios.

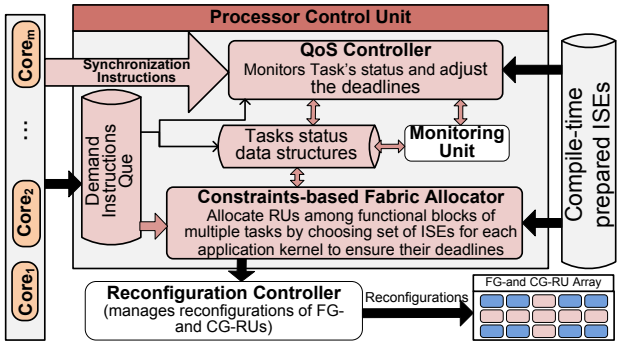

Fig. 2 : Overview of our Processor Control Unit

Fig. 2 shows the architectural overview of our proposed Processor Control Unit showing its two main components, i.e, the Constraints-based Fabric Allocator and the QoS Controller. Each task requests the fabric area for its upcoming predicted functional blocks by issuing a *Demand Instruction* to the processor control unit. The Processor Control Unit queues these *Demand Instructions* if it is already processing a previous *Demand Instructions*, otherwise it starts the Constraints-based Fabric Allocator to allocate the Reconfigurable Units (RUs) to the demanding tasks.

The goal of Processor Control Unit at run time is to meet the deadlines of each task. It employs the Constraints-based Fabric Allocator and the QoS Controller to achieve this goal. The QoS Controller is triggered by *Demand Instructions* and *Synchronization Instructions* that are issued at QoS check points. It determines the criticalities for all executing tasks, which is used as an input to Constraints-based Fabric Allocator.

***QoS Controller*:** The main objective of the QoS Controller at the prediction of functional blocks is to fine-tune the deadlines assigned to each functional block. Initially the deadlines are proportionally distributed at functional block level based on their expected execution time. At run-time the QoS Controller uses the real time values monitored by the monitoring unit and calculates the actual time taken by the previous functional blocks of the given task during the same QoS period. Based on the actual time t_k taken by pre-

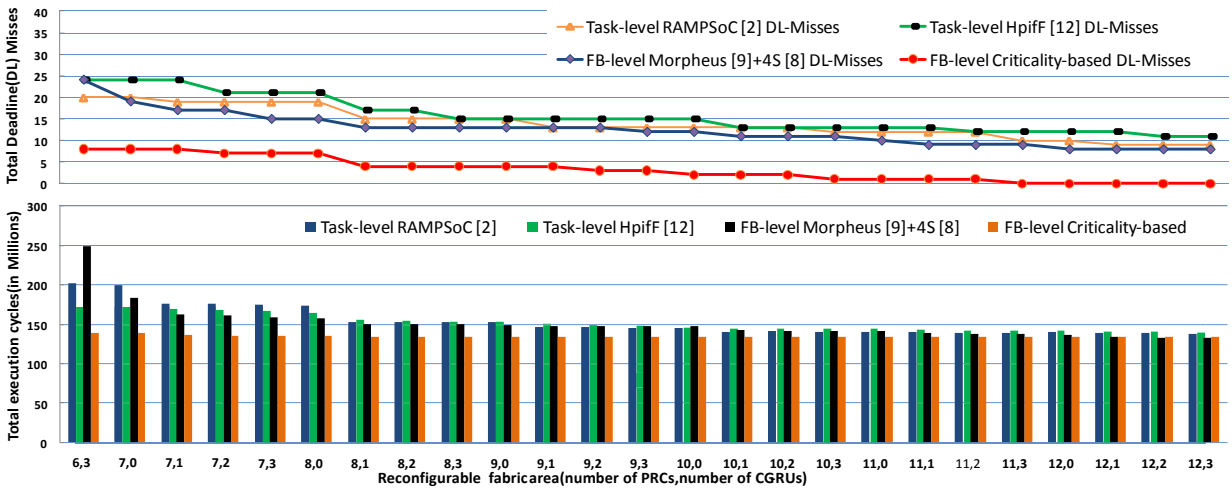

Fig. 3: A Comparison of performance and deadline misses with various state-of-the-art allocation policies

ceding functional blocks (i.e. *0,…,i-1*) and deadlines (*fb_dl*) assigned to the subsequent functional blocks (i.e. *i,…, n*), the QoS Controller estimates the total time that task would require to complete its job and then adjusts the deadlines of remaining functional blocks proportionally as explained in Eq. 1.

$$fb\_dl_i = fb\_dl_i - \left(\left(\sum_{k=0}^{i-1} t_k + \sum_{k=i}^{n} fb\_dl_k - \text{Task\_Deadline} \right) * \frac{\sum_{k=i}^{n} fb\_dl_k}{fb\_dl_i} \right) \quad (1)$$

The objective of the QoS controller at the completion of QoS period is to estimate the achieved QoS of each task and assign criticality. The criticalities are proportionally estimated based on the time left for each active task to complete its job.

Constraints-based Fabric Allocator: It allocates the RUs to the predicted functional blocks in such a way that each functional block can fulfill its deadline by choosing a performance efficient set of ISEs. The number of CG-RUs and the total number of PRCs are known to the Constraints-based Fabric Allocator. The task of the Constraints-based Fabric Allocator is to find the set of ISEs for all parallel executing tasks with goal that each functional block achieves its deadline while considering the constraints that the fabric used by all tasks at any instance is not more than the total fabric area. The output from this stage is the list of data-paths which need to be reconfigured on the allocated RUs. To find the best solution, we need to evaluate all combinations of ISEs for all tasks. Considering the run-time nature of our Constraints-based Fabric Allocator we incorporate a heuristic based algorithm.

Our approach first sorts the tasks with respect to their criticalities and favors the local optimal task t_s by giving more resources to it with the goal of minimizing the difference between the deadline $d_{x,y}$ and actual time $t_{x,y}$ taken by functional block y of task x as described in Eq. 2 (note that the functional blocks *fb* and *y* are limited to future subsequent blocks).

$$\text{find } t_s : \exists fb : t_{s,fb} - d_{s,fb} = \min_{\forall x,y} \{ t_{x,y} - d_{x,y} \} \quad (2)$$

The QoS Controller updated the task criticalities as explained before, which are feed back to Constraints-based Fabric Allocator to fine-tune the found solution.

V. Experimental Results and Evaluation

We used Leon-II processor (based on SPARC V8 architecture) as RISC cores. The CG-RUs are operating at 400 MHz, while the FG-RUs are operating at 100 MHz. The reconfiguration bandwidth of the FG-RU is 67584 KB/s (for Xilinx Virtex-II FPGAs). The 32-bit load/store unit is available to each CG-RU, while each FG-RU is provided with two 128-bit load/store units. A point-to-point communication between the RUs takes two cycles. The complete system is simulated on our multi-core cycle-accurate instruction-set-simulator. Its inputs (i.e., the data-paths latencies and reconfiguration cycles for FG- and CG-RUs) are obtained after place-and-route using Xilinx-FPGA-tools [3] and ASIC-synthesis-flow for TSMC using the same technology node (i.e., 90nm), respectively.

For evaluation, we use a secure video conferencing application which consists of H.264 video encoder/decoder and AES encrypt/decrypt tasks. The H.264 decoder and AES decrypt belong to one dependent set of tasks, while the H.264 encoder and AES encrypt belong to another set of dependent tasks. Due to their diverse processing behavior, these tasks vary in their fabric demands and corresponding performance improvement. For a fair comparison, we have provided the same set of hardware accelerators and ISEs as well as the same amount of memory bandwidth and same input data for each comparison.

A. Comparison with State-of-the-Art

We compare our scheme with state-of-the-art multi-core reconfigurable processors (like 4S [8], Morpheus [9]) and different task and functional block level allocation policies [13]. Fig. 3 shows a detailed comparison with state-of-the-art for the total execution time and the deadline misses. The horizontal axis shows the fabric area which is in pairs of number of PRCs and number of CG-RUs. The bars show the total execution time of an application and the lines show the dead-

line misses for the given area and allocation policy used. We discuss these comparisons as follows:

Compared to Task-Level RAMPSoC [2]: The comparison with RAMPSoC [2] is analogous to the comparison with the Equal distribution policy as all the cores have the same amount of fabric. Our proposed allocation scheme reduces (on avg.) 6.1x deadline misses and achieve up to 1.6x performance improvement. The major advantage in performance improvement is achieved when available fabric is limited, i.e., between (6,3) and (11,3). The reason is that due to equal fabric allocation to all tasks, the tasks requiring more fabric than others, suffers from deadline misses. In this scenario, the H.264 encoder and decoder miss most of their deadlines as they do not get the required fabric.

Functional block-level policy for Morpheus [9] and 4S [8] like architectures (FB-level Morpheus+4S): The architectures like Morpheus [9] and 4S [8] have proposed fabric allocation at compile-time for performance optimization. We have modeled their approach for resource allocation at run-time. Due to loosely coupled architecture, only the CG- or FG- ISE of each functional block could be used. Compared to their approach, our scheme provides up to 1.9x performance improvement and reduces the average deadline misses by 5.3x times. The advantages shown in the results is basically due to the fact that we consider the task criticality and adjust accordingly during run-time at functional block level, while Morpheus+4S like approach tries to achieve high performance without considering the criticality of each task.

Compared to Task-level Highest performance improvement factor First (HpifF [12]) policy: This comparison is analogous to comparing with techniques of [12] as they allocate the resources for overall performance optimization. Compared to [12], our scheme reduces (on avg.) 6.9x deadline misses and achieve up to 1.4x time performance improvements (avg. 1.3x). In HpifF based scheme, the task with the most potential of improvement factor (*pif)* is given more fabric. The *pif* is estimated cycles saved compared to non-accelerated execution of functional blocks. The problem with such an allocation scheme is that the task with the highest *pif* monopolizes the available resources and will always get the demanded resources. This is shown in this scenario as the H.264 encoder is using almost 80% of the fabric area in these cases; causing AES encrypt and decrypt tasks to completely miss their deadlines. On the contrary, our scheme reduces deadline misses by 6.9x times, as our processor control unit uses the actual monitored criticalities of tasks to allocate the fabric.

B. Overhead and Implementation Details

The Processor Control Unit uses one of the available CG-RUs to execute its algorithms. Constraints-based Fabric Allocator is the most compute intensive part. Its processing depends on number of tasks, number of *Demand Instructions*, number of kernels, size of fabric (no. of CG-RUs and PRCs), and number of data-paths that must be replaced for given already reconfigured data-paths. On average, in our experiments, the Constraints-based Fabric Allocator requires less than 1200 cycles for each task and on average it took 4500 cycles to decide the fabric share for 4 tasks that had put their *Demand Instructions*. The overhead is negligible as it is 1.8% of an average execution time of a functional block.

VI. CONCLUSIONS

Our novel Processor Control Unit enables efficient simultaneous multi-tasking in multi-core reconfigurable processors with functional block level allocation of the mixed-grained reconfigurable fabric, while maintaining the application's QoS requirement under run-time varying scenarios. Our Constraints-based Fabric-Allocator adaptively allocates the fabric to different tasks considering their refined criticalities, such that the margins for deadline misses are reduced. The QoS controller increases or decreases the criticality of each task based on the current state of the system and further guides the Constraints-based Fabric-allocator to refine its next allocation decisions. Compared to state-of-the-art, our scheme reduces the deadline misses by (on average) 6x, while improving the overall performance by 1.3x.

ACKNOWLEDGMENT

We thank German Research Foundation (DFG) for funding the work within the KAHRISMA project.

REFERENCES

[1] R. Koenig et al., "KAHRISMA: A Novel Hypermorphic Reconfigurable-Instruction-Set Multi-grained-Array Architecture", DATE, pp.819-824, 2010.

[2] D. Göhringer et al., "Runtime adaptive multi-processor system-on-chip: RAMPSoC", IPDPS, pp. 1-7, 2008.

[3] http://www.xilinx.com/products/v4q/lx.htm

[4] A. Bhattacharjee et al., "Thread Criticality Predictors for Dynamic Performance Power, and Resource Management in Chip Multiprocessors", ISCA, pp. 290-301, 2009.

[5] O. Diessel et al., "Dynamic scheduling of tasks on partially reconfigurable FPGAs", IEEE Proceedings – Computers and Digital Techniques, pp. 181-188, 2000.

[6] E. El-Araby et al., "Virtualizing and sharing reconfigurable resources in High-Performance Reconfigurable Computing systems", HPRCTA, pp. 1-8, 2008.

[7] A. Gavrilovska et al., "High-Performance Hypervisor Architectures: Virtualization in HPC Systems", HPCVirt, 2007.

[8] G. Smit et al., "Overview of the 4S project", In International Symposium on System-on-Chip, pp. 70–73, 2005.

[9] F. Thoma et al., "Morpheus: Heterogeneous reconfigurable computing", in FPL, pp. 409–414, 2007.

[10] E. Lübbers and M. Platzner, "ReconOS: An RTOS supporting hard- and software threads", in FPL, pp.441-446, 2007.

[11] K. Wu et al., "MT-ADRES: Multithreading on Coarse-Grained Reconfigurable Architecture", in International Journal of Electronics, Volume 95, Issue 7, pp. 761-776, 2008.

[12] W. Ahmed et al., "mRTS: Run-Time System for Reconfigurable Processors with Multi-Grained Instruction-Set Extensions", DATE, pp.1554-1559, 2011.

[13] C. Haung and F. Vahid, "Dynamic Coprocessor Management for FPGA-Enhanced Compute Platforms", CASES, pp.71-78, 2008.

[14] V. Gupta et al., "Self-adaptive admission control policies for resource-sharing systems", SIGMETRICS, pp. 311–322, 2009.

[15] L. Bauer et al., "Run-time System for an Extensible Embedded Processor with Dynamic Instruction Set", DATE, pp.752-757, 2008.