Hardware / Software Virtualization for the Reconfigurable Multicore Platform

Invited paper to the 10th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC 2012)

M. Ferger, M. Al Kadi, M. Hübner
Ruhr-University of Bochum (RUB)
Bochum, Germany
{max.ferger, muhammed.alkadi, michael.huebner}@rub.de

M. Koedam, S. Sinha, K. Goossens
Technical University of Eindhoven
Eindhoven, The Netherlands
{m.l.p.j.koedam,k.g.w.goossens, s.sinha}@tue.nl

G. Marchesan Almeida, J. Rodrigo Azambuja, J. Becker
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
{gabriel.almeida, jose.azambuja, becker}@kit.edu

***Abstract*—This paper presents the FlexTiles approach for the virtualization of hardware and software for a reconfigurable multicore architecture. The approach enables the virtualization of a dynamic tile-based hardware architecture consisting of processing tiles connected via a network-on-chip and a reconfigurable FPGA-based layer for application acceleration.**

Keywords—Multicore System-on-Chip; hardware and software virtualization; reconfigurable computing

I. Introduction and Motivation

Last years have seen the emergence of multicore and soon enough many-core systems as an answer to the ever-growing demand for more and more efficient computational power. Knowledge and tools to optimize programs for relatively small systems – say a few 1'000 to 100'000 lines of code – exist and are nearly perfected. Minimizing the underlying architecture in terms of size, power or cost can be considered well-understood; a wealth of workable solutions exists, whenever the optimizers – either humans or assistive algorithms – are presented with a more-or-less monolithic problem. One may observe that these constrained problems taken separately do not explain the growth of systems. It is their conglomeration onto a single platform and their intertwined usage by interactive and impatient users that explain the high pressure from vendors to build systems of ever higher performance and responsiveness, yet consuming less power and occupying less space in the device.

Part of the difficulty to specify an embedded system in an efficient way is caused by the unforeseeable mixture of applications and their different demands at runtime.

The power efficiency is dealt with by accelerating suitable parts of the applications through special processors like DSPs or specially configured hardware. Partially re-configurable FPGAs offer an adaptive platform to be optimized until the start of a task. This however needs knowledge about the future of the whole system:

- Is the accelerated part really the hot-spot when considering the actual amount of data to process?
- What if another program becomes more important?
- How can we predict that the long setup time of an FPGA based accelerator will pay out?
- If we successfully manage a high throughput in a very constrained region of the die, how do we cope with local overheating?
- Are we able to continue working, if we detect that some areas have been damaged?
- How is it possible to switch off parts on-the-fly?

The FlexTiles project aims at solving these issues in a pragmatic way, shown in Fig. 1. The main ideas are:

- to **divide** the whole mass of computational power into manageable parts or *tiles;*
- to abstract these tiles through **virtualization**, giving a high degree of *flexibility*; and
- to exploit the flexible tiles by means of **self-adaptation** in order to answer the individual challenges of an embedded platform at runtime.

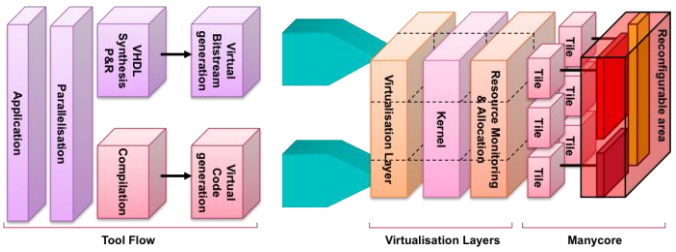

Figure 1. Overview of the FlexTiles Tool Chain and Tile Mapping

II. Related Work

To virtualize hardware, a composable and predictable multi-processor system-on-chip (MPSoC) platform is proposed in [1]. Composability ensures that applications cannot affect one another by even a single clock cycle. Hardware composability is achieved by making every shared resource composable. Networks-on-a-Chip (NoC), as proposed in [6] and [7], offer virtual streaming connections with guaranteed maximum latency and minimum throughput. Composable

This research is co-funded by the European Union under the Seventh Framework Programme within the FlexTiles project (FP7 ICT-288248).

processor virtualization is explained in [8]. A composable Real Time Operating System (RTOS) is explained in [5].

III. CONCEPT

A. Hardware

This section introduces the hardware architecture of the FlexTiles platform. It can be defined as a multicore heterogeneous platform. An overview of the hardware architecture is explained in Fig. 2. It is composed as a set of resources including both programmable General Purpose Processor (GPP) and Digital Signal Processor (DSP) cores, and reconfigurable dedicated hardware accelerators (HW-Acc). The hardware accelerators are mapped onto an FPGA fabric to provide a high level of flexibility. All the resources are connected through a NoC.

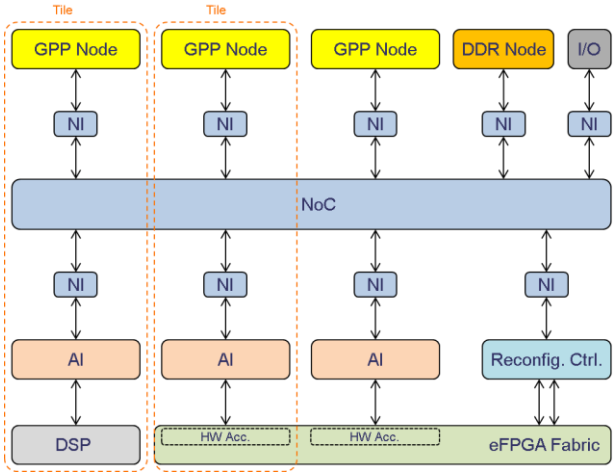

Figure 2. Architecture Overview (Communication Channels)

In the proposed execution model, GPP nodes are used as masters that can benefit from accelerators to delegate processing tasks. The accelerators, working as slaves, are either DSP cores or hardware accelerators mapped onto the eFPGA (embedded FPGA) fabric. To homogenize the control of accelerators by the GPP masters, a common Accelerator Interface (AI) is proposed.

In this architecture, a Tile is defined as a group of nodes including a single GPP and one or more accelerator nodes (DSP or HW-Acc mapped onto the FPGA fabric).

For the FlexTiles platform two NoC implementations will be evaluated: dAElite [7] and ANoC [9]. The dAElite NoC offers guaranteed services (GS) such as uncorrupted, lossless and ordered data delivery, guaranteed throughput, and bounded latency. It uses TDMA, where the time is divided into slots that are globally synchronized. Application connections can be set up or removed at runtime [10]. Application connections can be allocated offline [11] or online [12].

B. Software

This section introduces the software layer proposed in the FlexTiles platform. The layer can be classified into three parts: (1) operating system; (2) information management system; and (3) virtualization layer in software.

1) Operating System

In order to allow several threads to be scheduled, we embed an operating system into the GPPs. The chosen operating system is based on the CompOSe [1] real-time operating system (RTOS), where each node present in the many-core architecture runs the same OS instance. CompOSe implements composable time-division multiplexing (TDM) between applications, where each one has its own model of computation, task scheduler [2] and power manager [3]. The behavior of each application is completely independent of, and unaffected by that of any other application. We envisage that CompOSe will be extended in such a way that applications can be started, stopped and migrated dynamically at runtime. These features will enable us to implement self-adaptive techniques that will take place at runtime in order to reduce power consumption, to improve application performance and to reduce temperature hotspots among other challenges.

2) Information Management

The FlexTiles project will adopt the model called MDA, proposed in [4]. This model comprehends three phases of adaption: (M) monitoring, (D) diagnosis and (A) action. In the first phase (M), each tile monitors a given number of metrics that drive an application-specific mapping policy. The collected information is then analyzed and diagnosed by the second phase (D). Depending on the diagnosis obtained from the second phase, an action must be taken. In this case, the third phase (A) decides to push or attract tasks, which results in parallelizing, respectively in serializing the corresponding tasks' execution. Each node can scale frequency up or down in order to deal with application requirements or to save power.

The adopted model runs in a cyclic manner, where the sequence of the three phases is executed endlessly to optimize and refine the resource utilization in the system over time. In the following, each phase will be described in detail.

M) Monitoring phase

In order to monitor the system and extract metrics, some information must be provided by the architecture to the virtualization layer. To define the scope of the architecture we propose implementing three monitoring systems:

- **Application performance:** software services implemented in the operating system periodically monitor the performance of different applications running onto the GPP while special hardware keeps track of application performance in the DSPs and hardware accelerators.
- **Workload from CPU, DSPs and HW-Accelerators:** the operating system running on each tile monitors its CPU usage and hardware monitors keep track of the current usage of DSPs and accelerators of each tile.
- **Temperature:** Dedicated hardware monitors the temperature of each tile as well as the temperature of the routers of the NoC.

Additionally, voltage and frequency might be considered as inputs to make efficient decisions at runtime.

D) Diagnosis phase

During the diagnosis phase conclusions are drawn based on the monitored information and the most relevant information is provided to the decision-making mechanism in the action phase. The diagnosis mechanism extracts instant and averaged platform behaviors such as: (1) CPU overload, (2) CPU under-load (inefficient use), (3) decrease of application performance, and (4) overheating of the tile

For information critical to the system, that requires a fast response from the virtualization layer, this phase can be bypassed. In such a case, the monitored information is passed straight on to the action phase. For example, whenever the temperature of the circuit is too high, the decision-making is activated directly and a low latency decision is taken. It is important to notice that under such scenarios, the decision must be taken as soon as possible in order to avoid damage to the system. For the non-critical information, the diagnosis phase is executed before the decision-making mechanism starts.

a) Action phase

In the action phase decisions are made based on the information obtained from both monitoring and diagnosis phases. This mechanism can take one of the following actions: (1) reduce processor frequency, due to CPU under-load or an overheating tile; (2) increase processor frequency, in order to meet application performance requirements; (3) migrate tasks to even out efficient utilization of all CPUs; or (4) re-allocate hardware blocks by using a suitable defragmentation approach when loading new applications.

3) Virtualization Layer Overview

Fig. 3 illustrates the architecture overview of the GPP comprising the CompOSe RTOS and the Virtualization Layer modeled as an application. The Virtualization Layer (VL) interacts with the operating system by means of an Application Programming Interface (API) with basic functions exposed to the user space. These functions can only be accessed via the virtualization layer and include monitoring information and resource management.

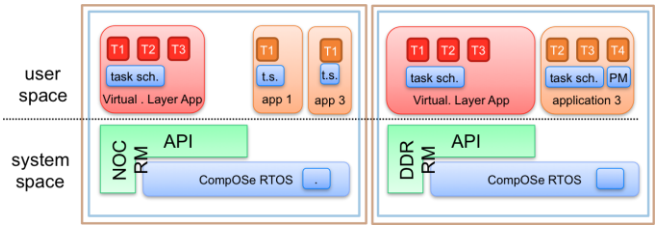

Figure 3. Architecture Overview (Virtualization Layer)

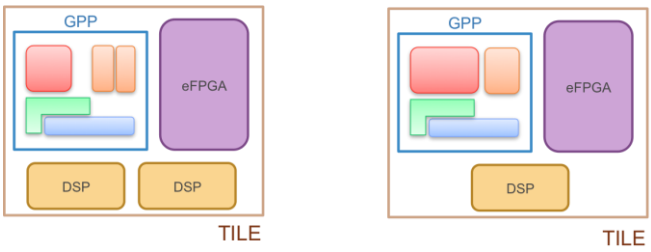

Figure 4. Different Configurations for the Tiles

The virtualization layer, embedded in a GPP, is modeled as an application and granted access to HW information. Fig. 4 depicts two different example configurations for the tile. The first one represents a tile composed of a GPP, an eFPGA layer and two DSPs, while the second one is composed of a single DSP. In FlexTiles, we adopted the concept of "virtual tile". With this approach, tiles may have different configurations with components managed by the local VL. The virtualization layer also manages the tile resources.

Fig. 5 illustrates the communication between the virtualization layer (VL) and CompOSe RTOS. The VL sends requests to the operating system and receives replies with the requested information.

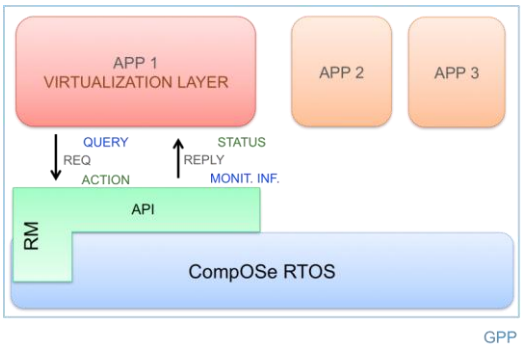

Figure 5. Communication between the Virtualization Layer and RTOS

There are two types of possible requests that the VL can execute: queries and actions. The VL might *query* for monitoring information at runtime, such as:

- energy consumption (per GPP, application or task);
- CPU utilization (overall, per application, per task);
- temperature of the GPP; or
- processor frequency.

The VL may request the execution of a *actions*, such as:

- remove an application or a task;
- load a new application;
- perform an application or task migration between tiles;
- pause an application or task on a GPP and load it onto a DSP or a HW-Acc in the eFPGA or vice-versa; and
- increase or decrease processor frequency.

Fig. 6 illustrates the distribution of the virtualization layer throughout the FlexTiles architecture. Each tile is intended to have different IP cores, such as hardware accelerators in the eFPGA, a GPP and additional DSPs.

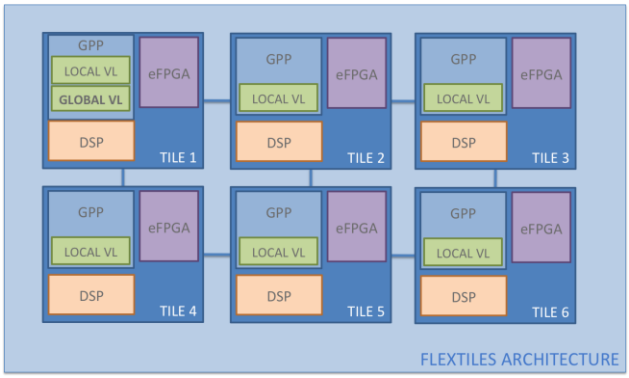

Figure 6. Distribution of Virtualization Layer over the FlexTiles Architecture

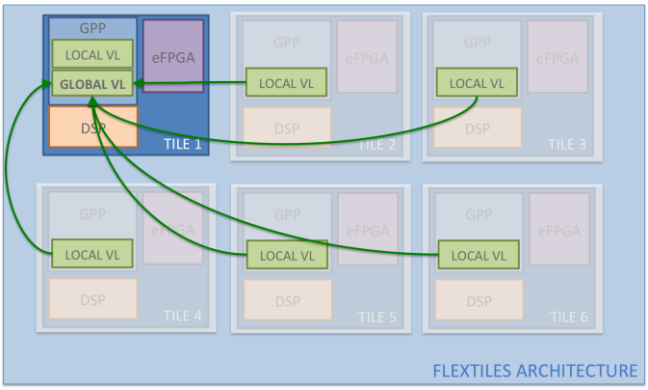

Figure 7. Communication between Virtualization Layers

Each tile in the architecture will have a local virtualization layer which will be responsible for making decisions at runtime and is allowed to make modifications (adaptations), but only to its own tile. It is not allowed to change any configuration in its neighborhood. The total number of local VLs running on the platform will be equal to the number of tiles chosen during the configuration process of the architecture. One additional, global VL allows some global optimization, taking into account the actual state of all tiles belonging to the system.

The idea of having local virtualization layers is that different adaptations can be executed in parallel (per tile) based on the monitored information gathered locally by each tile. This reduces the transmission of control packets over the network-on-chip in comparison to a concept with only one global VL. Summarized monitoring information is reported from all local VLs to the global VL by transmitting a few control packets. The communication between local VLs and the global VL is handled via an API, which allows the transmission of packets over a network-on-chip. These two approaches give more flexibility to the architecture when deciding e.g. which runtime techniques to be used in order to optimize for certain criteria like improving application performance or reducing overall power consumption.

Fig. 7 illustrates the scenario where Local VLs are reporting their monitored information to the Global VL, hosted in a master tile of the system.

IV. Conclusion

The FlexTiles project provides a hardware setup as well as an approach for the virtualization of a reconfigurable multicore hardware. The holistic concept and the realization provide a synergy between a local and a global virtualization approach and enable the usage of complex, dynamic and adaptable hardware from a high level of abstraction.

References

[1] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, “CoMPSoC: A template for composable and predictable multi-processor system on chips,” ACM Transactions on Design Automation of Electronic Systems, vol. 14, no. 1, pp. 1–24, 2009.

[2] A. Molnos, A. Beyranvand Nejad, B. T. Nguyen, S. Cotofana, and K. Goossens, “Decoupled inter- and intra-application scheduling for composable and robust embedded MPSoC platforms,” in MAP2MPSOC, May 2012.

[3] A. Nelson, A. Molnos, and K. Goossens, “Composable power management with energy and power budgets per application,” in SAMOS, Jul.2011.

[4] Gabriel Marchesan Almeida. *Adaptive Multiprocessor Systems-on-Chip Architectures: Principles, Methods and Tools*. ISBN 978-3848424283, 124 pages, LAP LAMBERT Academic Publishing (March 14, 2012), Germany.

[5] Andreas Hansson, Marcus Ekerhult, Anca Molnos, Aleksandar Milutinovic, Andrew Nelson, Jude Ambrose, and Kees Goossens. 2011. Design and implementation of an operating system for composable processor sharing. Microprocess. Microsyst. 35, 2 (March 2011), 246-260.

[6] A. Hansson, M. Subbaraman, and K. Goossens, “aelite: A flit-synchronous network on chip with composable and predictable services,” in Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE), Apr. 2009.

[7] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens, “A TDM NoC supporting QoS, multicast, and fast connection set-up,” in Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE), Mar. 2012.

[8] A.Molnos, A. Milutinovic, D. She, and K. Goossens, “Composable Processor Virtualization for Embedded Systems”. In: Computer Architecture and Operating System Co-Design, 2010.

[9] Y. Thonnart, P. Vivet, and F. Clermidy, “A fully-asynchronous low-power framework for GALS NoC integration,” in Design, Automation Test in Europe Conference Exhibition (DATE), 2010, march 2010, pp. 33 –38.

[10] A. Hansson and K. Goossens, “Trade-offs in the configuration of a network on chip for multiple use-cases,” in Proc. Int’l Symposium on Networks on Chip (NOCS), May 2007

[11] Andreas Hansson, Kees Goossens, and Andrei Radulescu. 2005. A unified approach to constrained mapping and routing on network-on-chip architectures. In Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS '05)

[12] R. Stefan, A. Beyranvand Nejad, and K. Goossens, “Online allocation for contention-free-routing NoCs,” in Proc. Interconnection Network Architecture: On-Chip, Multi-Chip (INA-OCMC), Jan. 2012.