Cool MPSoC Programming

Rainer Leupers
RWTH Aachen University, leupers@iss.rwth-aachen.de

Lothar Thiele
ETH Zurich, thiele@ethz.ch

Xiaoning Nie
Infineon Technologies, xiaoning.nie@infineon.com

Bart Kienhuis
Compaan Design, kienhuis@compaandesign.com

Matthias Weiss
Blue Wonder Communications, matthias.weiss@bluwo.com

Tsuyoshi Isshiki
Tokyo Institute of Technology, isshiki@vlsi.ss.titech.ac.jp

***Abstract*—This paper summarizes a special session on multi-core/multi-processor system-on-chip (MPSoC) programming challenges. Wireless multimedia terminals are among the key drivers for MPSoC platform evolution. Heterogeneous multi-processor architectures achieve high performance and can lead to a significant reduction in energy consumption for this class of applications. However, just designing energy efficient hardware is not enough. Programming models and tools for efficient MPSoC programming are equally important to ensure optimum platform utilization. Unfortunately, this discipline is still in its infancy, which endangers the return on investment for MPSoC architecture designs. On one hand there is a need for maintaining and gradually porting a large amount of legacy code to MPSoCs. On the other hand, special C language extensions for parallel programming as well as adapted process network programming models provide a great opportunity to completely rethink the traditional sequential programming paradigm for sake of higher efficiency and productivity. MPSoC programming is more than just code parallelisation, though. Besides energy efficiency, limited and specialized processing resources, and real-time constraints also growing software complexity and mapping of simultaneous applications need to be taken into account. We analyze the programming methodology requirements for heterogeneous MPSoC platforms and outline new approaches.**

I. INTRODUCTION

The best way of programming embedded MPSoC platforms, in terms of productivity and quality of results, is currently a topic for intensive discussion. This paper intends to contribute to this discussion by looking at various problem aspects, including general MPSoC architectural choices and their impact on programmability, the requirements of particular application domains, as well as programming models and corresponding tools support. Section II presents arguments for heterogeneous instead of homogeneous MPSoC platforms and derives design tool requirements. Section III suggests a concrete MPSoC architecture and programming model for future LTE platforms. In section IV, we outline the MAPS framework for mapping simultaneous embedded applications onto heterogeneous MPSoC. Section V discusses a combined MPSoC SW synthesis and simulation flow with application to streaming data processing. The tool presented in section VI is also targeted to streaming applications and provides a solution to auto-parallelization of certain sequential input programs into a Kahn Process Network (KPN) representation. Finally, the TCT programming model and tools framework presented in section VII exemplifies how a parallelizing compiler can support fine-grained MPSoC architecture exploration. From these case studies we can derive that the key to really "cool" MPSoC programming lies in creating tool workbenches that exploit knowledge about application and platform characteristics, rather than aiming for a push-button, "one-size-fits-all" MPSoC compiler approach.

II. MPSOC PROGRAMMING FOR ENERGY EFFICIENCY

Consider the energy equation

$$E = \Sigma[\, N(P0,1)+N(P0,2)+\ldots+N(P0,k)]*C(P0)*V^2$$

for a homogeneous multi-core cluster P0[1..k] where the same k cores P0 are used in the cluster, given the number of the clock cycles N on each core and given a design with effective capacitance C and operating voltage V. We can lower energy consumption by dynamic voltage scaling (DVS) as E is proportional to V^2. In fact, it is possible to schedule tasks in a way such that the consumed energy is minimized to meet deadlines [4][5].

To achieve further energy reduction we focus on MPSoC for a specific application domain such as mobile baseband processing. We take the MPSoC as a system consisting of software and multi-core hardware to perform a set of specific functions within a defined deadline, with certain N(P0,i) and certain C(P0) associated with the core P0[i] for i=1..k. Now we profile the task on a particular P0[i] further and design another specific instruction set processor core P1, such that we obtain N(P1,i) and C(P1) with N(P1,i) < N(P0,i) and C(P1) < C(P0). By replacing P0 with P1, we expect to achieve significantly higher energy efficiency for the application (class). This approach results in a heterogeneous MPSoC platform, actually

978-3-9810801-6-2/DATE10 © 2010 EDAA

with not only better energy efficiency but also with higher silicon cost efficiency [6][7].

However, the more efficient architecture does not come for free. In fact we need to consider another system parameter, namely the overall system reliability, which must be maintained at least as high as it is with a single core implementation. A particular challenge is to design software for a given reliability at reasonable design time and effort. In other words, the speed of the progress regarding productivity of software programming will largely determine the degree of parallelizing (heterogeneous) hardware platforms.

Today we have certain support e.g. from fast virtual prototyping techniques, application specific processor design as well as code generation tools for architecture exploration and software programming. This enables a gradual introduction of reliable heterogeneous MPSoC platforms. However there is still a great need for more effective methodologies, description languages, and tools. They are required especially for

- optimal task partitioning onto heterogeneous clusters of multi-core with respect to meeting deadlines and minimizing energy consumption.
- effective and correct-by-construction code generation tools not only for iterative architecture exploration, but also for software production and software maintenance.
- seamless integrated simulation and debugging environment, allowing e.g. virtual prototyping for software production, software debugging and hardware-software co-verification.

III. Optimizing and Programming MPSoC Architectures for Second Generation LTE Baseband Designs

First generation LTE baseband implementations were mainly targeted to cope with changing standards and early field tests. This was accomplished by either employing powerful FPGA platforms [8] or heavy Software Defined Radio (SDR) architectures [9]. Now, for the second generation architectures emphasis has to be put on lowest power consumption and smallest die size. The straightforward approach would be a fully hardwired implementation. However, significant flexibility is still required, e.g. for optimizations in the field. Hence, MPSoC architectures need to be employed for second generation LTE baseband system, too. In addition, new programming paradigms need to be introduced to allow for cost and power optimized implementations while still providing sufficient flexibility.

For second generation implementations significant parts of the standard are fixed and can be safely put into hardware. The main requirement for software programmability comes from the need to allow for adoptions in the field and – often more importantly – to optimize the system towards lowest power consumption. The BWC200 LTE architecture depicted in Fig. 1 addresses these needs by splitting the system into several independent units which are independently controlled by RISC processors and, where needed, supplemented by DSP processors.

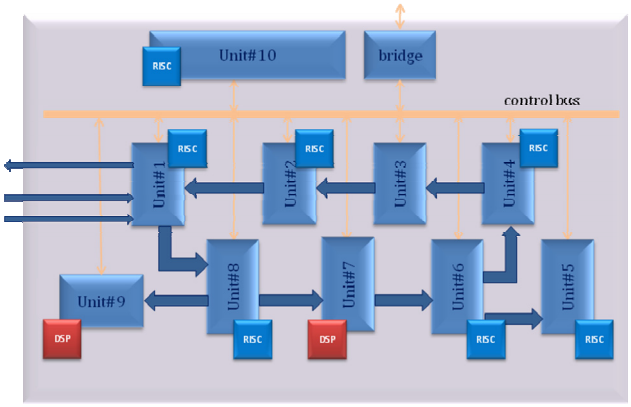

Fig. 1: The BWC200 MPSoC data stream architecture

These RISC and DSP processors are implemented based on Tensilica's Xtensa technology forming an MPSoC architecture. Its heterogeneous nature - due to the usage of different core configurations - demand for new principles in the MPSoC programming model, which are based on the following pre-requisites:

- Usage of processor cores of the same base architecture to allow the application of a unified control scheme.
- Employment of a data-streaming architecture to avoid heavy multi-layer bus systems.
- The usage of message based software programming models to abstract the details of the architecture from the programmer.

The programming model for this architecture makes use of the distributed nature of processing nodes. The programmer issues messages that can be interpreted by each core and hence the specific nature of each core configuration can be abstracted.

Triggered by a message a unit can receive a burst of data and conducts the processing either via hardware accelerators or in case of DSPs via firmware. Once this processing is complete it indicates to the succeeding unit via interrupts the availability of data. This receiving unit calls the data burst once it finalized its active task. By employing this data streaming communication the continuous processing of data can be achieved and blockings avoided.

This MPSoC programming model also allows for an energy efficient implementation by providing independent islands that can be target to local power optimizations and still fulfilling LTE's stringent real time constrains. Last, this approach provides the basis for multi-application programming or retargetable MPSoC software design flows as described in the sequel.

IV. TOWARDS MPSOC MULTI-APPLICATION PROGRAMMING

The transition from classical compilers for single-processor platforms to highly parallel MPSoCs poses several key challenges:

- **Multi-application compilation**: The functionality of wireless terminals more and more resembles that of traditional PCs. Multiple applications are activated dynamically and possibly simultaneously. This concerns application layer processing such as multimedia codecs as well as PHY layers of various radio standards. In most of today´s platforms both layers are separated and the PHY scheduling is rather static. However, this may change in the future for sake of better resource utilization and due to SDR and cognitive radio being on the roadmaps for future terminals. Due to real-time constraints for some applications, it is no longer sufficient to compile different applications separately. The compiler has to take various time-critical multi-application scenarios into account, in order to ensure that enough processing bandwidth is available even in worst case workload situations.

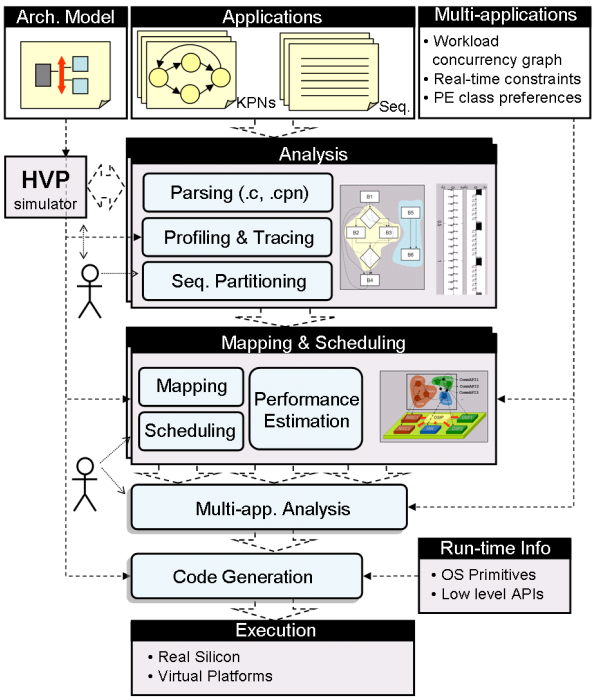

Fig. 2: Workflow in the MAPS compiler

- **Real-time constraints**: Traditional compilers are timing agnostic. However, embedded MPSoC applications fall into three major real-time classes: hard real-time (HRT, e.g. for digital receivers), soft real-time (SRT, e.g. video decoders), and non real-time (NRT, e.g. JPEG compression). Both the compiler for single applications as well as the global MPSoC scheduler have to be aware of these constraints for an optimal quality of service.
- **Code partitioning**: Due to the legacy code problem there cannot be a sudden dramatic change in programming principles and languages. We can rather expect a gradual change towards truly parallel programming, which will take significant time. Therefore, MPSoC compilers should be equipped with –at least semi-automatic- code partitioning functionality, capable of distributing applications given as sequential legacy code among the MPSoC processing elements.
- **Heterogeneity**: Due to energy efficiency and performance reasons, the processing elements in embedded MPSoCs are quite heterogeneous. Moreover, there are HW accelerators for certain functionalities, e.g. dedicated graphics engines. Programming formalisms for such platforms must expose this heterogeneity to a certain extent to the programmer and must be able to make corresponding optimized choices concerning task-to-processor assignment, code partitioning, and on-demand task migration.

The MAPS compiler project (fig. 2) at RWTH Aachen (as part of the UMIC Excellence Cluster) aims at tackling some of these challenges. MAPS accepts as input a set of (optionally) time-constrained applications. These can be given as either annotated sequential code or in a KPN notation based on extensions of C. Possible dynamic workload scenarios for HRT and SRT applications are captured in a concurrency graph, which allows to make static scheduling decisions. NRT applications are scheduled only dynamically in best-effort manner. A stand-alone high-level virtual platform (HVP) allows for a fast early SW performance estimation and workload distribution. In the core MAPS compiler, sequential code applications can then be semi-automatically split into tasks, which are translated together with KPN applications into a unified intermediate representation (IR). Next, spatial and temporal mapping of tasks to heterogeneous processing elements takes place, using profiling based performance estimation. Finally, a retargetable code generation phase maps the scheduled IR tasks into platform specific source codes and task communication primitives. Besides various virtual platforms, MAPS currently targets TCT (see section VII) and TI´s OMAP platform.

Further details on MAPS can be found in [1,2,3]. Another important future item in the MPSoC tool R&D roadmap is multiprocessor SW debugging. The classical SW debugging approach does not scale well for manycore architectures. We believe that new, abstract SW process oriented (rather than HW processor oriented) debugging methodologies and tooling are required to manage future SW complexity levels.

V. A RETARGETABLE MPSOC SOFTWARE DESIGN FLOW FOR STREAMING APPLICATIONS

A. *High Level Compilation*

In the following, we will discuss mainly application domains in which the MPSoC has to process a set of streaming applications, i.e. applications that can be partitioned into components that communicate with each other via data streams, see e.g. [14]. Increasingly, we are faced with sets of streaming applications that appear and disappear dynamically. As a result, even simple static stream-oriented applications will suffer from sporadic timing failures or may be subject to dynamic remapping because of overload or overheating situations.

Fig. 3 shows a possible software design process, see e.g. [11]. It is characterized by a relatively slow exploration cycle shown at the bottom of the chart: Starting from an architecture and application specification as well as from a mapping of software components to platform elements, at first optimized hardware-dependent software (HdS) components and operating system links are generated. They replace the conventional middleware approach by a more efficient compile-time generation which considers the application mapping. The compiled software will then be simulated on a virtual platform in order to validate functional correctness and non-functional properties. The results can be used to change the application and/or the mapping. In case of dynamic applications, some of these changes may involve adapting the online resource (re-)allocation strategy.

The growing complexity of applications and size of MPSoC platforms make it necessary to add a second loop shown at the top of fig. 3. The whole process of HdS generation, low-level compilation and simulation is replaced by analytic performance estimations that scale to large and complex applications. Advance methods based on component-based concepts for the real-time analysis of distributed embedded systems have successfully be applied in this context and embedded into a complete design flow, see e.g. [12], [13].

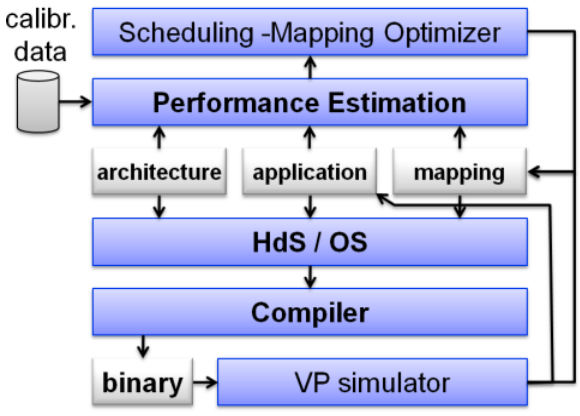

Figure 3: MPSoC high level compilation flow.

B. *Components and Sensitivity*

The core of analytic performance analysis is composability with respect to adding application components, adding resources as well as resource sharing strategies. This way, a fixed set calibrated local platform and application parameters can be used to estimate the global end-to-end system properties for a large set of possible application mappings.

Unfortunately, this concept of a small set of fixed local properties (determined via calibration [12]) that can be used to compute global system-wide characteristics is in danger in case of increased system complexity related to size and dynamic behavior. For example, a fixed WCET (worst case execution time) of a software process may not be an appropriate abstraction of the accumulated run-time of a process, if we are faced with caches, dynamic pipelines and interference on communication paths to memory. Timing estimations based on simple interference models may yield a vast loss in accuracy in case of high interference from other running applications. On the other hand, a more detailed analysis or even simulation will suffer from non-acceptable run-time. The above mentioned cross-interference between dynamic applications will lead to an increased non-determinism in the timing behavior, see also [15].

Possible new concepts and design guidelines to avoid this threat in terms of loss in accuracy in performance estimation and system predictability can be summarized as follows:

- Reduce the self-interference between several components of one single application and cross-interference between applications on common resource. This may be achieved through proper platform design or the use of resource servers that partition the available resources.
- Convince platform designers to select micro-architecture components whose behavior in terms of resource interaction can be modeled accurately on a high level of abstraction.
- Increase the robustness and sensitivity of hardware/software components such that non-deterministic input behavior is not amplified but possibly reduced, for appropriate definitions see e.g. [16]. Examples are the use of traffic shapers or scheduling servers.

This way, interactions on shared resources are reduced as well as more predictable such that relatively simple, component-based estimation methods can be applied effectively. This is a prerequisite for both, dynamic on-line resource management approaches with performance guarantees and future software design processes applicable to large-scale MPSoCs.

VI. FROM SINGLE THREADED C-CODE TO MULTI THREADED MPSOC CODE FOR HIGH PERFORMANCE STREAMING APPLICATIONS

A big problem with the arrival of multicore platforms is that most software is still written in a sequential style, typically in C code in the embedded space, focusing on a single processor with large global memory. Multicore

platforms on the other hand have multiple processors and typically a distributed, hierarchically organized memory. Therefore, the most used programming style conflicts in two dimensions; single thread program versus multiple thread programs and shared memory versus distributed memory. Nevertheless, designers need a trajectory to get them quickly from the current programming model to the programming model that fits with multicore design. This trajectory should happen as smoothly as possible as designers have to work with legacy code of possibly thousand lines of code. Having to validate the correctness of the new multithreaded program is not an option.

Compaan Design realizes this trajectory by the flow shown in fig. 4, which is based on the technology described in [17]. This flow helps a designer in two steps towards a multithreaded design while maintaining a correct working application consisting of possibly many thousand lines of code. In the original C code, a designer indicates with the Compaan On/Off pragma one or more regions. On each region a special auto-parallelization step is done (step 1) based on exact dataflow analysis. The result of this step is expressed as a KPN, i.e., a network of processes exchanging data using FIFOs (First In First Out buffers). Once a designer has such a KPN, the mapping (step 2) onto an MPSoC is relatively straightforward; a process becomes either a software thread on a microprocessor or a hardware accelerator. A FIFO connection could be map to native FIFO support but also to shared memory, a bus structure, or on a Network on a Chip (NoC) infrastructure.

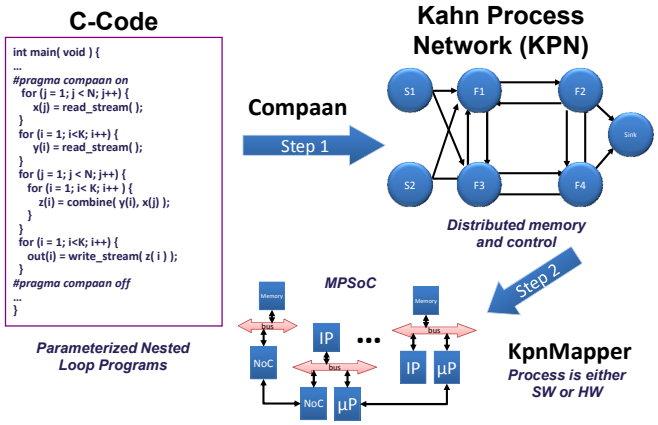

Figure 4: The MPSoC programming flow of Compaan Design

C-code is translated into an equivalent KPN representation using mathematical techniques based on the polytope model. This model is a geometric representation of the original computation expressed in C-code. For the polytope model very powerful mathematical manipulation techniques exist that are exploited to do the translation. Since the translation is mathematical without the use of heuristics, there is always a one-to-one relation between the C-code and the resulting KPN; as a result no validation is further needed. The use of the polytope model does limit the translation to a specific class of programs called Parametric Nested Loop Programs. However, any Matrix-Matrix or Matrix-Vector program can be expressed in this way and most signal processing applications typically consist of these kinds of operations. Allowing dynamic C-code constructs is subject to further research.

Compaan can also generate an indefinite number of different KPNs, all with the same input-output behavior. Each KPN has different characteristics, i.e., numbers of channels and processes. This can be viewed as expressing the C code at different degrees of parallelism. Using a special *merging* technique, Compaan can reduce the number of processes and thereby reducing the level of parallelism. Compaan can also do the reverse, by *splitting* a process in new processes, thereby increasing parallelism. By repeating this process in both directions, one can obtain many different partitions, covering the full range from no parallelism to full parallelism, all for a single piece of C-code.

VII. The Tightly Coupled Thread Model: Efficiently Exploring the MPSoC Design Space

Developing highly optimized MPSoC application programs requires enormous efforts that need to be finely tuned at the algorithm level, system partitioning, and mapping on the target MPSoC architecture. Maintaining the correctness of the parallelized code during these software refinements itself is difficult if the programmers are held responsible for preventing deadlocks and race conditions, and ensuring the correct execution order. The Tightly-Coupled Thread (TCT) Model developed at Tokyo Institute of Technology addresses this issue of MPSoC programming by providing a simple programming model on top of the sequential C code where the

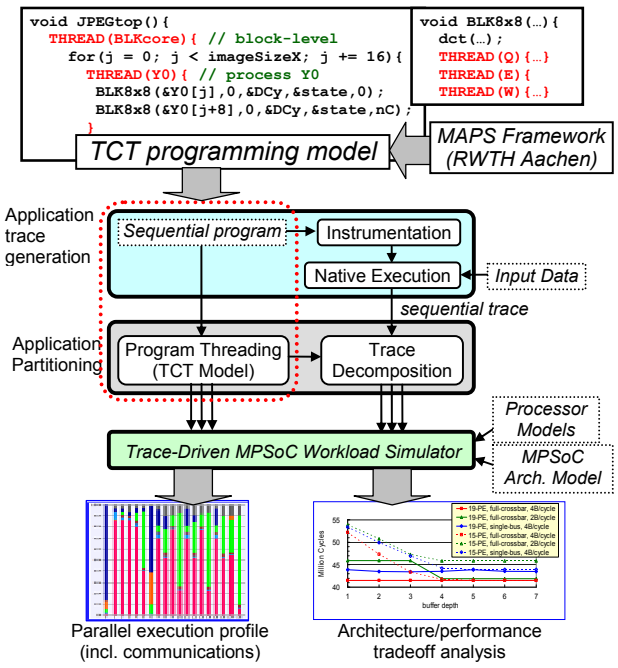

Figure 5: MPSoC programming and performance estimation framework on the TCT Model

programmers insert "thread scopes" in the code to describe the slicing structure of the application [18]. Our TCT compiler equipped with powerful interprocedural dependence flow

analysis is able to extract all dependences between these threads (which may reside in different functions) and automatically insert communication and synchronization instructions on the partitioned parallel codes. Thus the programmers are freed from maintaining the correctness of the parallelized code especially on the communication details and are allowed to directly focus on algorithm refinement and code restructuring for revealing more parallelism on the target MPSoC architecture. The MAPS framework from RWTH Aachen [1] includes powerful dataflow analysis and code partitioning engines that can emit thread-annotated code from sequential C code to be fed to our TCT compiler.

The underlying communication model in the TCT Model is that of the message-passing protocol for distributed memory systems where each data dependence between thread pairs is converted into a DT (data transfer) instruction at the source thread and a DS (data synchronization) instruction at the destination thread. In addition to these DT/DS instructions, CT (control token) instructions are added for activating the inner control-dependent threads. These three primitive communication instructions with several additional instructions for maintaining the communication buffers are automatically generated by the TCT compiler, resulting in a behavior that is both data-driven (by DT/DS) and control-driven (by CT) to ensure a correct execution order of parallelized codes. An efficient instruction-driven message-passing scheme has been developed during the design of our TCT-MPSoC prototype chip [19] which requires very short setup time of 2 to 6 cycles and high bandwidth data transfer of 4 bytes/cycle assisted by the communication model embedded in the processor element. Our TCT-MPSoC implementation has also been ported to commercial ESL tool environments to realize the TCT-MPSoC virtual platform composed of processor models and crossbar-based interconnect models.

For early stage MPSoC architecture exploration and application tuning, we have also developed a new MPSoC performance estimation framework based on trace-driven workload models [20]. This framework, illustrated in fig. 5, includes an automatic fine-grain workload model generation of SW components (coded on TCT Model), application trace generation through automatic source-level instrumentation using a new program trace encoding technique, and a MPSoC workload simulator kernel which, in addition to generating the accurate timing behavior of processor elements, reproduces the inter-processor communication traffic on a parameterized MPSoC interconnect model. Our trace-driven MPSoC workload simulator is capable of quickly estimating the performance of a wide range of architectural parameters of MPSoC such as the number of processors, interconnect topology, communication bandwidth and buffer size.

VIII. REFERENCES

[1] J. Ceng, J. Castrillon, W. Sheng, H. Scharwaechter, R. Leupers, G. Ascheid, H. Meyr, T. Isshiki, H. Kunieda: MAPS: An Integrated Framework for MPSoC Application Parallelization, 45th Design Automation Conference (DAC), Jun 2008

[2] R. Leupers, J. Castrillon: MPSoC Programming using the MAPS Compiler, 15th Asia and South Pacific Design Automation Conference (ASPDAC), Jan 2010

[3] J. Castrillon, R. Velasquez, A. Stulova et al.: Trace based Composability Analysis for Mapping Simultaneous Applications to MPSoC Platforms, Design, Automation, and Test in Europe (DATE), Mar 2010

[4] Wanghong Yuan, Klara Nahrstedt, Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. SOSP'03, October 19–22, 2003, Bolton Landing, New York, USA.

[5] F. Gruian. Hard real-time scheduling for low energy using stochastic data and DVS processors. Intl. Symp. on Low-Power Electronics and Design, Aug. 2001.

[6] X. Nie, U. Nordqvist, L. Gazsi, D. Liu : Network processors for access network (NP4AN): trends and challenges. IEEE International SOC Conference, Spet. 2004.

[7] Intel® Technology Journal: Tera-scale Computing . Volume 11, Issue 03 ,http://www.intel.com/technology/itj/2007/v11i3/4-environment/2-intro.htm

[8] Chanbok Jeong et all: An efficient UE modem platform architecture for 3GPP LTE , Proceeding of the SDR 06 Technical Conference, 2006

[9] K. Moermann: Embedded Vector Processors Hold Key to Software-defined Radio, Wireless Design&Development, ASIA, Sep 2008

[10] Fanny Mlinarsky: Multimode wireless devices: It's the software, stupid!, Mobile Handset design Online, Nov 2009

[11] W. Haid, K. Huang, I. Bacivarov, and L. Thiele: Multiprocessor SoC Software Design Flows. IEEE Signal Processing Magazine, vol. 26, no. 6, pp. 64—71, Nov. 2009.

[12] W. Haid, M. Keller, K. Huang, I. Bacivarov, and L. Thiele: Generation and Calibration of Compositional Performance Analysis Models for Multi-Processor Systems. In Proc. Int'l Conf. on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), pages 92—99, Samos, Greece, July 2009.

[13] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse: System architecture evaluation using modular performance analysis: A case study. Int. J. Softw. Tools Technol. Transfer, vol. 8, no. 6, pp. 649–667, Nov. 2006.

[14] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli: Design of embedded systems: Formal models, validation, and synthesis. Proc. IEEE, vol. 85, no. 3, pp. 366–390, Mar. 1997.

[15] Lothar Thiele, Reinhard Wilhelm: Design for Timing Predictability. Real-Time Systems, Vol. 28, No. 2, pages 157-177, 2004.

[16] R. Racu, M. Jersak, R. Ernst: Applying sensitivity analysis in real-time distributed systems. Real Time and Embedded Technology and Applications Symposium (RTAS) 2005, pages 160- 169, March 2005.

[17] Bart Kienhuis, Edwin Rijpkema, and Ed F. Deprettere "Compaan: Deriving Process Networks from Matlab for Embedded Signal Processing Architectures.", 8th International Workshop on Hardware/Software Codesign (CODES'2000)

[18] M. Z. Urfianto, T. Isshiki, A. U. Khan, D. Li, H. Kunieda, Decomposition of Task-Level Concurrency on C Programs Applied to the Design of Multiprocessor SoC, IEICE Trans. 91-A(7), pp.1748-1756, 2008

[19] M. Z. Urfianto, T. Isshiki, A. U. Khan, D. Li, H. Kunieda, A Multiprocessor SoC Architecture with Efficient Communication Infrastructure and Advanced Compiler Support for Easy Application Development, IEICE Trans. 91-A(4), pp.1185-1196, 2008

[20] T. Isshiki, D. Li, H. Kunieda, T. Isomura, and K. Satou, Trace-Driven Workload Simulation Method for Multiprocessor System-On-Chips, In Proc. DAC 2009