SESAM Extension For Fast MPSoC Architectural Exploration And Dynamic Streaming Applications

N. Ventroux, T. Sassolas, R. David, G. Blanc, A. Guerre and C. Bechara

CEA, LIST,
Embedded Computing Laboratory
91191 Gif-sur-Yvette CEDEX, France;
Email: nicolas.ventroux@cea.fr

Abstract—**Future systems will have to support multiple and concurrent dynamic compute-intensive applications, while respecting real-time and energy consumption constraints. To overcome these computation needs, only multithreaded approaches are possible. Thus, the support of a streaming execution model is very important for dataflow applications. However, with dynamic applications, each execution stage is prone to execution time variations. The sizing of these highly complex MPSoC architectures becomes difficult. In such a context, flexible and accurate simulators become a necessity for exploring the vast design space solution. In this paper, we use the SESAM environment to ease the architectural exploration of asymmetric MPSoCs for dynamic streaming application processing. This paper focuses on the new programming and execution model supported by the simulator, and studies performances obtained with a WCDMA encoder/decoder application implemented on a complete MPSoC platform.**

I. Introduction

The emergence of new embedded applications for telecom, automotive, digital television and multimedia applications, has fueled the demand for architectures with higher performances, more chip area and power efficiency. These applications are usually computation-intensive, which prevents them from being executed by general-purpose processors. Architectures must be able to simultaneously manage concurrent information flows; and they must all be efficiently dispatched and processed. This is only feasible in a multithreaded execution environment. Designers are thus showing interest in System-on-Chip (SoC) paradigms composed of multiple computation resources and networks that are highly efficient in terms of latency and bandwidth. The resulting new trend in architectural design is the MultiProcessor SoC (MPSoC) [1].

Another very important feature of future embedded computation-intensive applications is the dynamism. Algorithms become highly data-dependent and their execution time depend on their input data, since decision processes must also be accelerated. Consequently, on a multiprocessor platform, optimal static partitioning cannot exist since all the processing times depend on the given data and are prone to non-uniform data accesses. In [2], it is shown that the solution consists in dynamically allocating tasks according to the availability of computing resources. Global scheduling should maintain a balanced system load and support workload variations that cannot be known off-line. Moreover, the preemption and migration of tasks dynamically balance the computation power between concurrent processes.

One possible approach to parallelize an application is to pipeline its execution. This programming and execution model suits well for data oriented applications that consider a continuous flow of data. An asymmetrical approach can implement a global scheduling and efficiently manage dynamic streaming applications. An asymmetric MPSoC owns a centralized control manager that handles the application execution, and can distribute the pipeline stages among computing resources.

In a previous work [3], we have developed the SESAM tool to help the design of new asymmetric MPSoC architectures. This tool allows the exploration of MPSoC architectures and the evaluation of many different features (effective performance, used bandwidth, system overheads...). In this paper, we extend SESAM to support streaming execution. With this important feature, SESAM becomes able to explore different MPSoC solutions and dataflow application implementations. With the study of a WCDMA encoder/decoder implementation on a complete asymmetric MPSoC architecture, we will validate our work and understand what impacts the performances and limits streaming executions.

This paper is organized as follows: Section II covers related works on MPSoC simulators from both industrial and academic worlds. Then, section III gives an overview of the initial SESAM environment. Section IV presents its programming model, while section V focuses on new SESAM's features to support pipelined dataflow applications, from the programming to the execution model point of view. Section VI illustrates the performance results obtained by running a real case embedded application on a complete MPSoC architecture implemented with SESAM. Finally, section VII concludes the paper by discussing the presented work.

II. Related Work

Lots of works have been published before on single-processor, multiprocessor and full-system simulators [4], [5]. Some of them focus on the exploration of specific resources. For instance, Flash [6] eases the exploration of different memory hierarchies, or SICOSYS [7] studies only different Network-on-Chips (NoCs). Taken separately, these tools are very interesting but a complete MPSoC exploration environment is needed in order to analyze all architectural aspects under real application processing case.

Among complete MPSoC simulators, MC-Sim [5] uses a variety of processors, memory hierachies or NoC configurations but remains cycle-accurate. On the contrary, simulators like STARSoC [8] offer a rapid design space exploration but only consider functional level communications. To study network contentions and the impact of communication latencies, a timed simulation is necessary. Others, like ReSP [9], use generic processors and cannot take into account instruction set specificities. This does not allow to size and to validate MPSoC architectures. On the contrary, some simulators, like MPARM [10], are processor specific and do not allow the exploration of different memory system architectures or different processors, and hence lacks flexibility. In addition, MPARM requires AMBA compatible IPs to be integrated in the design, which requires the development of specific wrappers.

Some of the simulators benefit from the genericity of a very high description level, like Sesame [11] or CH-MPSoC [12]. They use a gradual refine Y-Chart methodology to explore the MPSoC design space. However, even if they remain very promising tools, they cannot support complex IPs or MPSoC structures with advanced networking solutions. Generated architectures remain very constrainted. Less generic projects exist, like SoCLib [13], but their scope are too limited to fulfil MPSoC exploration and in particular they cannot support automatic MPSoC generation to analyze its parameters.

Some very interesting projects [14], [15] make a model of a large set of MPSoC platforms. Nonetheless, these solutions do not propose a rich set of Network-on-Chips (NoCs), and it is not possible to easily integrate a centralized element to dynamically allocate tasks to resources. The programming model consists in statically allocating threads onto processors, and does not allow the design of architectures optimized for dynamic applications.

Finally, to the authors' knowledge, there is no published work on a simulator that supports asymmetric MPSoC architectures and allows MPSoC exploration for dataflow and dynamic applications. Simulating a whole MPSoC platform requires to find an adequate trade-off between simulation speed and timing accuracy.

III. SESAM OVERVIEW

SESAM is a tool that has been specifically built up to ease the design and the exploration of asymmetric multiprocessor architectures [3]. This tool is made of a set of instruction set simulators (MIPS, PowerPC, Sparc), networks-on-chips (multibus, mesh, torus, multistage, ring), a DMA, a memory management unit, caches, memories and different control solutions to schedule and dispatch tasks. We can also study the pipeline length impact of processing elements [16]. All the simulators provided blocks can be timed.

SESAM can also be used to analyze and optimize the application parallelism, as well as control management policies. This framework is described with the SystemC description language, and allows MPSoC exploration at the TLM level with fast and cycle-accurate simulations. It supports co-simulation within the ModelSim environment [17] and takes part in the MPSoC design flow, since all the components are described at different hardware abstraction levels.

Besides, SESAM uses approximate-timed TLM with explicit time to provide a fast and accurate simulation of highly complex architectures. This model, described in [18] uses the Transactional Level Modeling (TLM) approach coupled with timed communications. This solution allows the exploration of MPSoCs while reflecting the accurate final design. Regarding the communications, we point out a 90 % accuracy compared to a fully cycle-accurate simulator. Time information is necessary to evaluate performances and to study communication needs and bottlenecks.

To ease the exploration of MPSoCs, all the components and system parameters are set at run-time from a parameter file without platform recompilation. It is possible to define the memory map, the applications that must be loaded, the number of processors and their type, the number of local memories, their size, the parameters of the instruction and data caches, memory latencies, network latencies, network topologies (torus, ring, mesh...) etc. More than 160 parameters can be modified. Moreover, each simulation brings more than 250 different platform statistics. That helps the designer size the architecture. For example, SESAM collects the miss rate of the caches, the memory allocation history, the processor occupation rate, the number of preemptions, the time spent to load or save the context of tasks, or the effective used bandwidth of each network.

A script can be used to automatically generate several simulations by varying different parameters in the parameter file. An Excel macro imports these statistics to study their impact on performances. In addition, SESAM offers the possibility to automatically dispatch all the simulations to different host PCs. For example, 400 simulations can be carried out with 12 hosts in less than one hour and a half [3].

Debugging the architecture is possible with a specific GNU GDB [19] implementation. In the case of a dynamic task allocation modeling, it is not possible to know off-line where a task will be executed. Therefore, we build up a hierarchical GDB stub that is instantiated at the beginning of the simulation. A GDB instance, using the remote protocol, sends specific debug commands to dynamically carry out breakpoints, watchpoints, as well as step by step execution, on an MPSoC platform. This unique multiprocessor debugger allows the task debugging even with dynamic migration between the cores. Moreover, it is possible to simultaneously debug the platform and the code executed by the processing resources.

IV. INITIAL PROGRAMMING MODEL OF SESAM

The programming model of SESAM is specifically adapted to dynamic applications and global scheduling methods. Obviously, it is inconceivable to carry out a generic programming model for all asymmetrical MPSoCs. Nonetheless, it is possible to add new programming models. The programming model is based on the explicit separation of the control and the computation parts. As depicted in Figure 1, each application must be manually (the tool chain is still under development) parallelized and cut into different tasks. Mainly, tasks are cut according to loop nests and dependencies must be explicitly expressed.

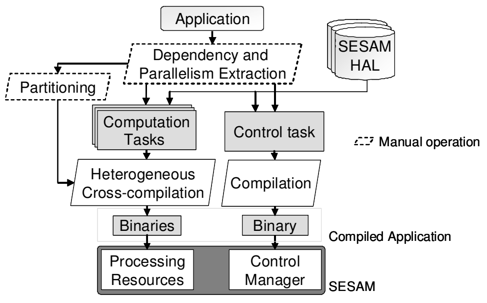

Figure 1. SESAM programming model

With this programming model, a specific Hardware Abstraction Layer (HAL) is provided to manage dynamic data memory allocations (Table I). It can be extended to explore other memory management strategies. This HAL provides memory allocation, read/write shared memory access and debugging functions. Each data is defined by a data identifier, which is used to dialog between the memory management unit and the computation tasks. For instance, the function call *sesam_data_assignation(10,4,2)* allocates *4* pages for the data ID *10* with *2* consumers for this data. The function call *sesam_write_data(10,&c,4)* writes the word *c* starting from the 4^{th} byte of the data ID *10*.

HAL functions	Description
Memory allocation functions	
sesam_reserve_data()	reserve pages for a future allocation
sesam_data_assignation()	allocate the data
sesam_free_data()	deallocate the data
sesam_chown_data()	change the owner of the data
Data access functions	
sesam_read(), sesam_write()	read or write the content of a data
sesam_read_burst()	read a finite number of bytes
sesam_write_burst()	write a finite number of bytes
sesam_read_byte()	read a byte
sesam_write_byte()	write a byte
Debug function	
sesam_printf()	display debug on a terminal

Table I
INITIAL HARDWARE ABSTRACTION LAYER OF SESAM

Each embedded application can be divided into a set of independent threads, from which explicit execution dependencies are extracted into a control task. Then, each thread can be divided into a set of finite computation tasks. A computation task is a standalone program. The greater the number of extracted independent and parallel tasks, the more the application can be accelerated at runtime. This acceleration comes at the expense of the control overhead. Then, a manual partitioning must be carried out in case of heterogeneous MPSoCs. Heterogeneous resource management takes place before the task compilation. Finally, all tasks are compiled for the processing resources or the control manager. Depending on the hardware platform to explore, the designer can use the SESAM HAL or explicit physical addresses without memory virtualization.

The control task is a Control Data Flow Graph (CDFG) extracted from the application, which represents all control and data dependencies. The control task handles the computation task scheduling and other control functionalities, like synchronizations and shared resource management. It can preempt and migrate tasks to balance computing resource's load between homogeneous resources. A dedicated and simple assembly language is used to describe this CDFG and must be manually written. Each control task, for each different application, needs to define: the number of computation tasks, the binary file names corresponding to these tasks, and their necessary stack memory size. Then, we must specify which are the first and last tasks of the application. We must also describe each CDFG transition. Finally, for real-time task scheduling, the deadline of the application, as well as the worst case execution time of each task, must be defined. The processor type of each task is also specified and this information is used during the allocation process. Only this representation of the control task is necessary for each available controller, whereas a specific compilation tool is used for the binary generation. Each task is defined by a task identifier, which is used to dialog between the control and the computation parts.

V. Streaming programming and execution model

The support of the streaming execution requires various modifications of the SESAM simulator. First of all, the programming model must be enriched with new functionalities to allow local synchronizations between tasks. Then, the memory management unit must be modified, as well as the kernel to support the scheduling of streaming applications.

With a streaming execution, the most important feature is the management of shared buffers between the pipeline stages. A consumer must wait for the shared data to be written before reading it, in order to keep coherent data. This requires the implementation of a specific protocol. Then, to maximize the parallelism in the pipeline and ensure sufficient concurrent executions, the granularity of data synchronizations must be well-sized. A fine-grain synchronization level generates an important hardware and control overhead to implement all semaphores used to store the access status information. For this reason, we decided to synchronize all shared data accesses at the page level. This is a simple and practical implementation of a streaming support, but remains different from direct handshaking works that have been done in [20]. Our synchronization mechanisms are less intrusive and not blocked by acknownledges. Besides, all this extra control will generate a task execution overhead that will have to be evaluated, in order to validate the efficiency of this execution model.

In the SESAM framework, the memory space can be implemented with several banks or a single memory. Memory segments are protected and reserved for the Control Manager. The Memory Management Unit (MMU) manages the memory space and shares it between all active tasks. To support all page synchronizations, we implemented two new functions named *sesam_wait_page* and *sesam_send_page* (Table II). The first function is a blocking wait method. The task waits for the disponibility of a page in only read or write mode. When all consumers have sent a write availability, the second function is used to inform the memory management unit that the content of the page is ready to be read, or that its content has become useless for the consumer task. The memory management unit can then release the page access rights and accept future writes, .

HAL functions	Description
Page synchronization functions	
sesam_wait_page()	wait for the availability of a page
sesam_send_page()	page is ready to be read or written

Table II
NEW HARDWARE ABSTRACTION LAYER FUNCTIONALITIES OF SESAM

When a *sesam_send_page* is sent to the MMU, the status of the page is updated. If the page was in a *write* mode, the consumer number is checked and updated. To distinguish multiple requests of a single task from multiple consumers' requests, a consumer list is maintained for each page. When all consumers have read the page, the page status changes and it becomes possible to write again into it. When a *sesam_wait_page* is sent to the MMU, the request is pushed into a *wait_dispo list request* and the information is sent to the controller. As soon as the page becomes available, the MMU sends to the processor an answer that unlocks the waiting *sesam_wait_page* function. Because a task can dynamically be preempted by the controller and migrated to another processing element, the MMU must be able to address the processor executing the waiting task. Thus, a *sesam_wait_page* is sent again when the task is resumed on the new processor in order to update the processing element address.

This hand-shake protocol is a semaphore-like processing and guarantees the data coherency. In addition, deadlocks are avoided since it is a dataflow model similar to Kahn Processes [21]. The main difference with KPNs is that synchronizations are centralized in order to bring more flexibility and allow the task migration. The MMU supports these functionalities through several control Finite State Machines (FSM). A status memory is implemented to store the read/write status, the producer task identifier, the processing element address and the list of consumers. A specific Content Adressable Memory (CAM) is used to address this status memory with the data identifier and the page number.

Finally, the kernel executed by the control processor has to be modified. When a task waits for the availability of a page, its status is changed to be preempted by another task. This prevents the inter-blocking and the under-utilization of processing resources, since only the effective execution of tasks is important. In addition, a priority level must be defined in the control task for all the pipeline's tasks in order to impose a priority to downstream tasks, i.e. tasks that are deeper in the pipeline have the priority over tasks that are found earlier in the pipeline. After the task execution analysis, this strategy shows a very important improvement in terms of preemptions, hence overall performances.

With all these new capabilities, the SESAM framework is able to carry out local control synchronizations, in order to let concurrent tasks share intermediate data. This new streaming execution model can be used to parallelize dataflow applications, and can be mixed with the initial constrained-task execution model. With both execution models, all the task parallelism techniques are supported in SESAM at the system level. Now, we have to analyze and quantitatively evaluate the control and task execution overhead due to the streaming execution, so that it becomes valuable.

VI. RESULTS

To demonstrate the SESAM's capabilities to support the execution of dataflow applications, we have used this framework to carry out a complete MPSoC architecture.

As shown in Figure 2, this architecture is made of homogeneous MIPS32 processing elements (MIPS32R2 ISA compliant functional ISSs) and two DMAs for the communication with the external data memory. These devices communicate with 64 shared memories through a 64-bit multibus data network, where each memory bank is 16KB. These memories are locally shared and physically distributed. The data network and memory latencies are 2 cycles. All the processors use two private instruction and data caches. Each cache is 2KB, 4-way set associative, and implements a write-back and write-allocate protocol. The external network-on-chip (NoC) is a 2-cycle-latency 32-bit simple bus with a round-robin arbiter. External memories have a latency of 4 cycles. All latencies have been evaluated through partial TSMC 45 nm ASIC synthesis and normalized in relation to the PE frequency. The central control manager is also a MIPS32 processor with two instruction and data caches. It executes a microkernel, which supports the dynamic scheduling and allocation of tasks on PEs. An interrupt controller is used to communicate with the computing resources. The architecture is also composed of a code loading unit (CLU) that can prefetch task instruction codes into shared memories, as well as the modified memory management unit (MMU), which has been previously presented. The memory space is cut into 256-byte pages. All devices are timed and only communications are approximate-timed transactions.

The implemented application is a complete Wide-band Code Division Multiple Access (WCDMA) encoder and decoder [22]. This communication technology is based on the use of Orthogonal Variable Spreading Factor (OVSF) to allow several transmitters to send information simultaneously over a single communication channel. This application uses a rake receiver with a data aided channel estimation method.

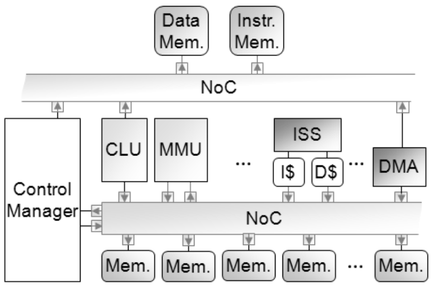

Figure 2. SESAM infrastructure

Known pilot symbols are transmitted among data. The channel estimation algorithm operates on the received signal along with its stored symbols to generate an estimate of the transmission channel. The processing of pilot frames generates a dynamic behavior of the application, since this induces a variable execution length.

The application is pipelined into 13 different tasks. To maximize the concurrency between pipelined tasks, a double buffer is used between each task. Thus, tasks can independently execute the next frame from the previous pipelined stage results. To study the acceleration and the benefits that could be obtained with a streaming execution, we compare in Figure 3 the execution of the non-parallelized application on a standalone PE, with a streaming implementation on several processors.

As shown in Figures 3-a and 3-b, the parallelism obtained with a streaming execution can be important. We get an acceleration of 4.5 and an occupation rate beyond 80 % with 8 PEs. The acceleration is constrained by the pipeline length, which is limited to 13 tasks. However, the task execution on a standalone processing element is penalized by more cache misses, since all the application shares the same caches. These results depend on the control overhead, which must be minimized. In our architecture, we use a microkernel especially developed to optimize the reactivity of the control. However, when the number of PEs increases (up to 16 PEs), the task execution and control overhead become predominant. This is mainly due to the control complexity that increases with the number of tasks and PEs to manage, and also because of the limited parallelism level of the application.

The streaming description of the application and the use of the streaming protocol to access shared data has also a non-negligible cost. Many accesses to a central device, such as the MMU, to get the authorization to write or read each page of a buffer, could have a very negative impact on performances. Only simulations help the evaluation of the potential benefit. In Figures 3-c and 3-d, the task execution overhead is represented and helps to understand the penalty induced by a streaming execution model. Without parallelization, the task execution overhead is insignificant and is only due to the task loading or saving. On the contrary, the task execution overhead must be taken into account within a streaming execution. It represents 22 % of the total execution time with 8 PEs. It is mainly due to page availability waits and this increases drastically with the number of processing elements. Regarding the contentions into the network, since we use a multibus network all PEs have their own access to each memory bank, and therefore only FIFOs on each memory bank can induce an additional latency. For instance, with 8 PEs the increase latency to access a memory bank is about 0.43% for data accesses and 0.12% for instruction accesses. These results are due to a smart allocation of data and instructions between memory banks by the MMU.

These results show interesting benefits when using a streaming execution. The parallelization is effective with many processors and turns out to be scalable if the application pipeline length is sufficient. The protocol used to access shared data has a negative impact on performances, but this execution model remains a good trade-off for multiprocessing execution. Nonetheless, the control overhead must be efficiently managed and all the optimization done in this study should not mask its importance.

VII. Conclusion

This paper presented an asymmetric MPSoC simulator named SESAM enhanced with new functionalities to support the streaming execution of dataflow applications. Asymmetric MPSoCs are adapted for the execution of dynamic embedded applications and can efficiently support the execution of a pipelined application on a limited number of computing resources. With memory management unit modifications and an extension of the hardware abstraction layer, a new streaming execution model is now supported into the SESAM framework. This contributes to considerably enlarge supported applications and allows new hybrid execution model studies.

The efficiency of this parallelism approach for dataflow applications were studied through the simulation of a complete MPSoC architecture. A WCDMA encoder and decoder application was implemented onto this platform. Interesting results have been obtained, thanks to the SESAM exploration environment. They demonstrated the performance of this execution model in spite of the induced control and synchronization overheads. An acceleration of 4.5 was reached with 8 processing elements.

Acknowledgements

Part of the research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement No. 100029.

References

[1] A. A. Jerraya and W. Wolf. *Multiprocessor Systems-on-Chips*. Elsevier, 2005.

[2] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566, April 2008.

[3] N. Ventroux, A. Guerre, T. Sassolas, L. Moutaoukil, C. Bechara, and R. David. SESAM: an MPSoC Simulation Environment for Dynamic Application Processing. In *IEEE International Conference on Embedded Software and Systems (ICESS)*, Bradford, UK, July 2010.

[4] J. J. Yi and D. J. Lilja. Simulation of computer architectures: simulators, benchmarks, methodologies, and recommendations. *IEEE Transactions on Computers*, 55(3):268–280, March 2006.

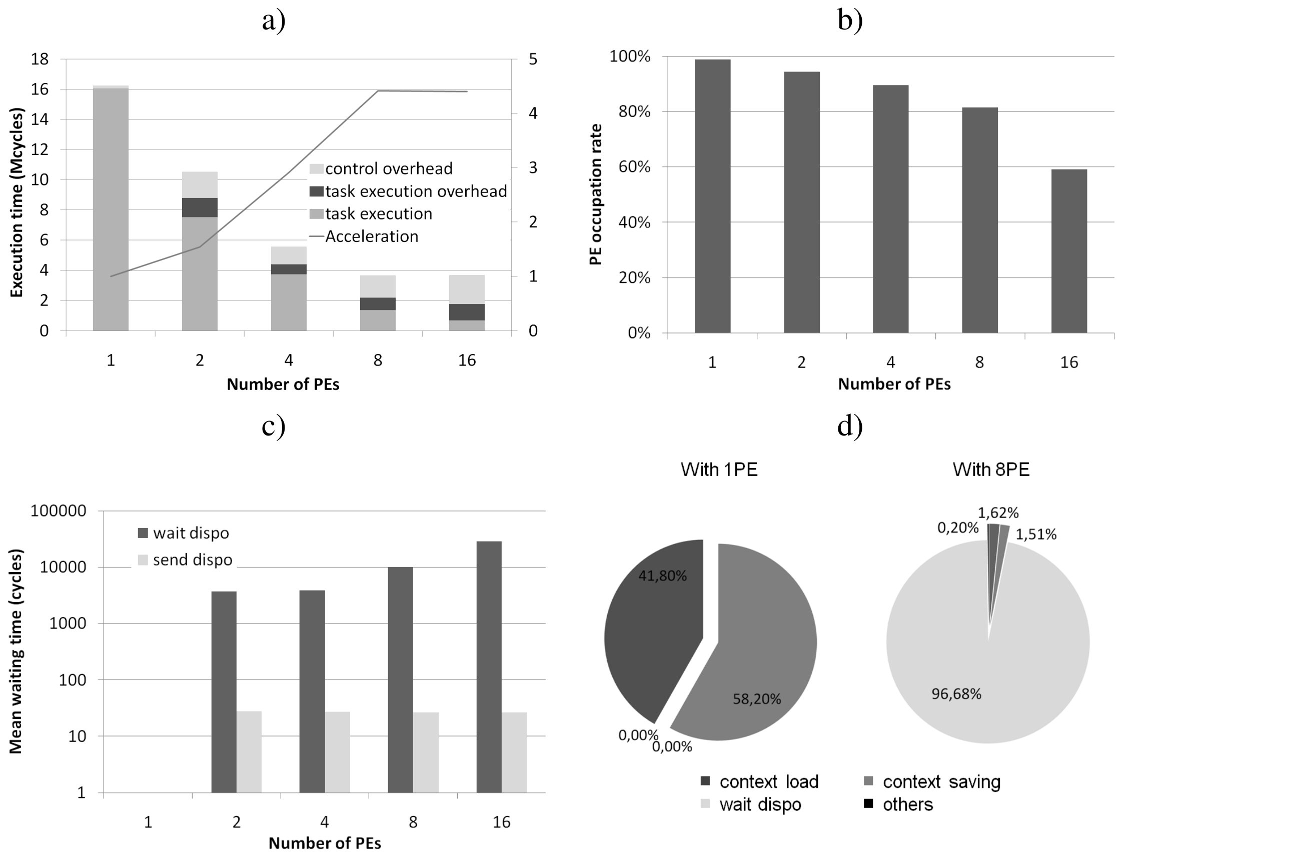

Figure 3. WCDMA encoder/decoder implementation results on a MPSoC platform with a streaming execution (except with one PE): (a) total execution time, task execution overhead and control overhead depending on the processing element (PE) number (MIPS32 processors); (b) utilization rate of PEs; (c) mean waiting time for the management of shared buffers; and (d) details of task execution overheads that take part in the task execution time with 1 and 8 PEs (with 1 PE the application is not pipelined)

[5] J. Cong, K. Gururaj, G. Han, A. Kaplan, M. Naik, and G. Reinman. MC-Sim: An efficient simulation tool for MPSoC designs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 364–371, San Jose, USA, November 2008.

[6] J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich. FLASH vs. (Simulated) FLASH: Closing the simulation loop. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Pittsburgh, USA, March 2000.

[7] V. Puente, J. Gregorio, and R. Beivide. SICOSYS: an integrated framework for studying interconnection network performance in multiprocessor systems. In *Euromicro Workshop on Parallel, Distributed and Network-based Processing*, Canary Islands, Spain, January 2002.

[8] S. Boukhechem and E.-B. Bouernnane. TLM Platform Based on SystemC For STARSoC Design Space Exploration. In *NASA/ESA Conference on Adaptive Hardware and Systems*, Noordwijk, The Netherlands, June 2008.

[9] G. Beltrame, C. Bolchini, L. Fossati, A. Miele, and D. Sciuto. ReSP: A non-intrusive Transaction-Level Reflective MPSoC Simulation Platform for design space exploration. In *Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 673–678, Seoul, Korea, January 2008.

[10] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. *VLSI Signal Processing Systems*, 41(2):169–182, 2005.

[11] A.D. Pimentel, C. Erbas, and S. Polstra. A Systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transactions on Computers*, 55(2):99–112, February 2006.

[12] H. Shen, P. Gerin, and F. Pétrot. Configurable Heterogeneous MPSoC Architecture Exploration Using Abstraction Levels. In *IEEE/IFIP International Symposium on Rapid System Prototyping*, Paris, France, June 2009.

[13] E. Viaud, F. Pêcheux, and A. Greiner. An efficient TLM/T modeling and simulation environment based on conservative parallel discrete event principles. In *DATE*, Nice, France, April 2009.

[14] A. Wieferink, T. Kogel, R. Leupers, G. Ascheid, and H. Meyr. A System Level Processor/Communication Co-Exploration Methodology for Multi-Processor System-on-Chip Platforms. In *International Conference on Design, Automation and Test in Europe (DATE)*, Paris, France, February 2004.

[15] P. Paulin, C. Pilkington, and E. Bensoudane. StepNP: A System-Level Exploration Platform for Network Processors. *IEEE Design & Test*, 19(6):17–26, November 2002.

[16] C. Bechara, N. Ventroux, and D. Etiemble. Towards a Parameterizable Cycle-Accurate ISS in ArchC. In *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*, Hammamet, Tunisia, May 2010.

[17] ModelSim. http://www.model.com/.

[18] A. Guerre, N. Ventroux, R. David, and A. Merigot. Approximate-Timed Transaction Level Modeling for MPSoC Exploration: a Network-on-Chip Case Study. In *Euromicro Conference on Digital System Design (DSD)*, Patras, Greece, August 2009.

[19] The GNU GDB project. http://www.gnu.org/software/gdb/.

[20] T. Henriksson and P. van der Wolf. TTL Hardware Interface: A High-Level Interface for Streaming Multiprocessor Architectures. In *IEEE/ACM/IFIP Workshop on Embedded Systems for Real Time Multimedia (ESTIMedia)*, Seoul, Korea, October 2006.

[21] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74*, North-Holland, USA, 1974.

[22] Andrew Richardson. *WCDMA Design Handbook*. Number ISBN 0521828155. 2006.