SESAM: an MPSoC Simulation Environment for Dynamic Application Processing

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

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. With the increase in the design complexity of MPSoC architectures that must support these constraints, flexible and accurate simulators become a necessity for exploring the vast design space solutions. In this paper, we present an asymmetric MPSoC simulator environment, named SESAM. This tool can be used for the architecture exploration and optimization, and the design of a complete MPSoC solution for dynamic application processing. Its performances and capabilities are demonstrated through a complete MPSoC platform and an implementation of the component labeling algorithm.**

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 process 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 a System-on-Chip (SoC) paradigm composed of multiple computation resources and a network that is highly efficient in terms of latency and bandwidth. The resulting new trend in architectural design is the MultiProcessor SoC (MPSoC) [1].

Another important feature of future embedded computation-intensive applications is the dynamism. Algorithms become highly data-dependent and their execution time depends on their input data, since decision processes and the whole application are now implemented and must be accelerated. Consequently, on a multiprocessor platform, optimal static partitioning cannot exist since all the processing times depend on the given data. [2] shows that the solution consists in dynamically allocating tasks according to the availability of computing resources. Global scheduling maintains the system load-balanced and supports workload variations that cannot be known off-line. Moreover, the preemption and migration of tasks balance the computation power between concurrent real-time processes. If a task has a higher priority level than another one, it must preempt the current task to guarantee its deadline. Besides, the preempted task must be able to migrate on another free computing resource to increase the performance of the architecture.

Unfortunately, existing architectures offer only partial solutions to the power, chip area, performance, reliability and dynamism problems associated with embedded systems. Only an asymmetrical approach can implement a global scheduling and efficiently manage dynamic applications. An asymmetric MPSoC owns a centralized control manager that manages the application execution.

Designing an MPSoC architecture requires the evaluation of many different features (effective performance, used bandwidth, system overheads...), and the architect needs to explore different solutions in order to find the best trade-off. In addition, he needs to validate specific synthesized components to tackle technological barriers. For these reasons, the whole burden lies on the MPSoC simulators, which should be parameterizable, fast and accurate, easily modifiable, support wide ranges of application specific IPs and integrate new ones easily.

In this context, we developed the SESAM tool, to help the design of new MPSoC architectures. The novelty of SESAM is its support to asymmetrical MPSoC architectures, which includes a centralized controller that manages the tasks for different types of computing resources. The heterogeneity can be used to accelerate specific processing, but the task migration is not supported. The best trade-off between the homogeneity, which provides the flexibility to execute dynamic applications, and the heterogeneity, which can speed-up the execution, can be defined in SESAM. Moreover, this tool enables the design of MPSoCs based on different execution models, which can be mixed, to find the best suitable architecture according to the application. In addition, SESAM can support simultaneous multiple different applications

and mix different abstraction levels, and can take part in a complete MPSoC design flow.

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 SESAM, as well as the supporting programming and debugging tools. Section IV describes SESAM's components and focuses on its infrastructure. Section V illustrates the performance results

obtained by running a real case embedded system application on a complete MPSoC architecture modeled with SESAM. Finally, section VI concludes the paper by discussing the presented work.

II. Related Work

Lot of works have been published before on single-processor, multiprocessor and full-system simulators. In [3], the authors illustrate a wide range of simulators, mainly targeting general-purpose computing. In a more recent work [4], Jason Cong et al presented an interesting classification of MPSoC simulators. We will mainly discuss three kinds of complete MPSoC simulators that exist in the literature and allow MPSoC exploration: ReSP [5], MPARM [6] and MC-Sim [4].

The ReSP simulator [5] runs in a Python environment using reflective capabilities [7] and instantiates SystemC IP cores with an automatic Python wrapper generation. This solution speeds up the design space exploration and eases the debugging environment. Their results do not show performance degradation with respect to a pure SystemC simulation environment. Though their solution is optimized for the reliability analysis, it is not suitable for co-design simulation. The reason is that the IP signals are wrapped by Python, hence they cannot be detected by ASIC and FPGA design tools.

MPARM [6] is a SystemC based modeling and simulation environment for symmetrical MPSoC architectures. It includes models for ARM processors, AMBA bus architecture, memory subsystems and multiprocessor synchronization modules. The main limitation of the simulator is its processing modules. Each module is composed of an ARM7 core, with its peripherals such as an interrupt controller, an UART, a timer, as well as tightly-coupled instruction and data caches. The processing module is written in C++ and encapsulated in a SystemC wrapper. This solution does not allow the exploration of different memory system architectures, and hence lacks flexibility. In addition, MPARM requires AMBA compatible IPs to be integrated in the design, which requires the development of specific wrappers.

Finally, MC-Sim [4] allows the simulation of a variety of NoC architectures, processor cores based on SESC [8], and L1 and L2 caches. In addition, MC-Sim has a novel approach to generate and integrate a cycle-true coprocessor models from C code in the MPSoC simulator. All these modules are flexibly configured to different positions in the interconnection network, in order to generate a variety of MPSoC topologies. However, like all existing MPSoC simulators, it is not possible to easily integrate a centralized element to allocate tasks to resources dynamically. The programming model consists in statically allocating threads onto processors, and do not allow the design of architectures optimized for dynamic applications.

For the authors' knowledge, there is no published work on a simulator that supports asymmetric heterogeneous MPSoC architectures and allows MPSoC exploration. Simulating a whole MPSoC platform needs to find an adequate trade-off between simulation speed and timing accuracy. The Transactional Level Modeling (TLM) approach coupled with timed communications, is a solution that allows the exploration of MPSoCs that reflects the accurate final design [9]. Time information is necessary to evaluate performances and to study communication needs and bottlenecks. Moreover, co-simulation is essential to refine the architecture and to design specific components into such a complex environment.

III. SESAM overview

SESAM is a tool that has been specifically built to ease up the design and the exploration of asymmetric multiprocessor architectures. It can 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 [10] and takes part in the MPSoC design flow, since all the components are described at different hardware abstraction levels.

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 name of the applications that must be loaded, the number of processors and their type, the number of local memories and their size, the parameters of the instruction and data caches, memory latencies, network types and latencies, etc. More than 120 parameters can be modified. Moreover, each simulation brings more than 200 different platform statistics, that help the architect sizing 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 task contexts, the effective used bandwidth of each network, etc. As depicted in Figure 1, 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. Thus, the cache parameters, the network bandwidths, as well as the effective performance of the architecture, are ones among many features that can be evaluated to size and explore MPSoCs.

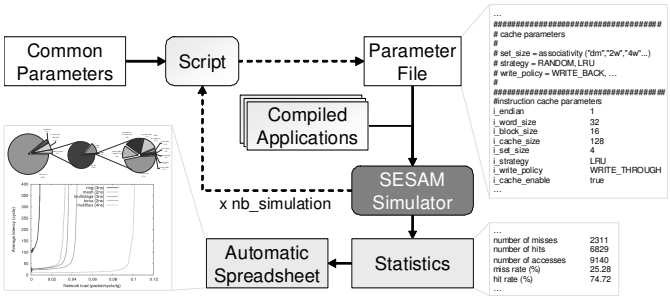


Figure 1. SESAM exploration tool and environment

Because the exploration of many parameters can take a lot of simulation time, SESAM offers the possibility to au-

tomatically dispatch all the simulations to different host PCs. Each available PC core defines an available slot, which can be used to execute one simulation. The tool is structured around a dispatcher and a NFS server (Figure 2). Thus, SESAM can take benefits of available PCs to automatically parallelize simulations and ease the exploration of architectures. For example, 400 simulations can be carried out with 12 hosts (40 slots) in less than one hour and a half to execute the model presented in section V, i.e. the whole architecture SCMP and the labeling application.

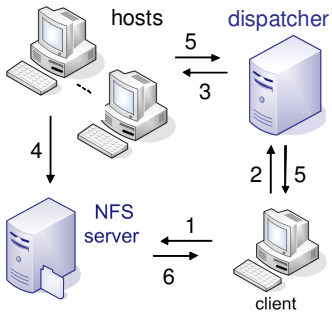


Figure 2. Parallelization of SESAM simulations. The client copies the simulator and its libraries on an NFS share folder (1). Then, the client sends to the dispatcher the list of simulations to be executed with their parameter files (2). The dispatcher looks for an available slot to execute each simulation (3). Then, the host changes its working directory to the NFS share folder and executes the simulation. At the end of each simulation, the result is written into the share folder (4), and the host informs the client through the dispatcher (5). Finally, the client gets back all the results (6).

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 proposed programming model is based on the explicit separation of the control and the computation parts. As depicted in Figure 3, each application must be manually (the tool chain is still under development) parallelized and cut into different tasks. Thus, computation tasks and the control task are extracted from the application, so that each task is a standalone program. The control task handles the computation task scheduling and other control functionalities, such as synchronizations and shared resource management. Each embedded application can be divided into a set of independent threads, from which explicit execution dependencies are extracted. Each thread can in turn be divided into a finite set of tasks. The greater the number of independent and parallel tasks are extracted, the more the application can be accelerated at runtime. A specific Hardware Abstraction Layer (HAL) is provided to manage all memory accesses and dynamic memory allocation. Others can be developed to explore different memory management strategies for example. Depending on the hardware platform to explore, the designer can use this library or explicit physical addresses without memory virtualization. Each task is defined by a task identifier, which is used to dialog between the control and the computation parts. 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 and made available to the processing resources or the control manager.

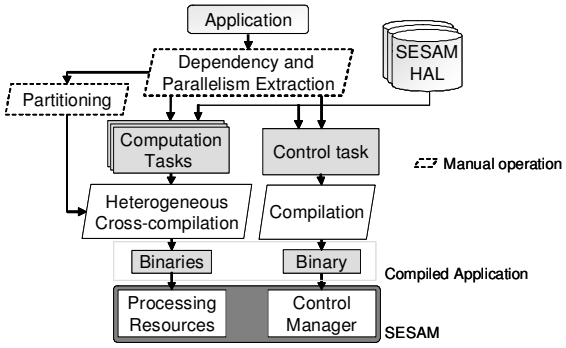


Figure 3. SESAM programming model

Debugging the architecture is possible with a specific GNU GDB [11] 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 built 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.

Besides, SESAM uses approximate-timed TLM with explicit time to provide a fast and accurate simulation of highly complex architectures. This model, described in [9], allows the exploration of MPSoCs while reflecting the accurate final design. 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. Thus, all provided blocks of the simulator are timed and the communications use a timed transactional protocol.

IV. SESAM INFRASTRUCTURE

As depicted in Figure 4, SESAM is structured as an asymmetrical MPSoC. It is based on a centralized Control Manager that manages the execution of tasks on processing elements. SESAM proposes the use of different components to design new MPSoCs. Other SystemC IPs can be designed and integrated into SESAM if they have a compatible TLM interface. The main elements are: the Memory Management Unit (MMU), the Code Loading Unit (CLU), Memories, a set of Instruction Set Simulators (ISS), a Direct Memory Access (DMA) unit, a Control Manager and Network-on-Chips (NoC).

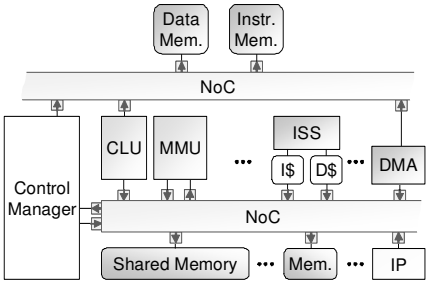


Figure 4. SESAM infrastructure

A. MMU

The MMU is optional and can bring advanced capabilities to manage all the shared memory space, which is cut into pages. The whole page handler unit is physically distributed between the MMU and the local Translation Lookaside Buffers (TLB) for each processing core. All the memory functions are available through the SESAM HAL. It is possible to dynamically allocate or deallocate buffers. There is one allocated buffer per data block. An identifier is used for each data block to address them through the MMU, but it is still possible to use physical addresses. Different memory allocation strategies are available and can be implemented.

B. CLU

The CLU dynamically loads task codes from the external memory through a DMA access when it receives a configuration command from the Control Manager. Then, in a dynamic memory management context, it also has to update the MMU to provide the corresponding virtual to physical address translations. A context and a stack are automatically included for each task.

C. Memory

Different memory elements can be instantiated. The memory space can be implemented as different banks or a single memory. The former is logically private or shared, while the latter is only shared between the processors. Memory segments are protected and reserved for the Control Manager. Multiple readers are possible and all the requests are managed by the NoC.

D. ISS

We use processors, designed with the ArchC language, as processing resources with data and instruction cache memories, which are optionals. The ArchC tool [12] generates functional or cycle-accurate ISS in SystemC with a TLM interface [13]. A new processor is designed in approximately 2 one-man weeks, but it depends on the instruction set complexity. Its simulation speed can reach tens of Millions of simulated Instructions Per Second (MIPS). Different models are available (Mips, PowerPC, Sparc), as well as a complete Mips32 processor (with a FPU) at the functional level. Preemption and migration of tasks are possible services that are available through an interruption mechanism. It allows to switch the context of the processing unit, to save it, and to restore the context code from the executed task memory space.

E. DMA

A DMA is necessary to transfer data between the external data memory and the internal memory space. A DMA is a standard processing resource and takes part in the heterogeneity of the architecture. It is a fully-programmable unit that executes a cross-compiled task for its architecture. A 3-dimensional DMA is available. Transfer parameters can afterwards be dynamically modified by other tasks, to specify source and target addresses defined at run-time. Finally, it dynamically allocates the required memory space for the transfer.

F. Control Manager

The Control Manager can be either a fully programmable ISS, a hardware component, or a mix of both. With the ISS, different algorithms can be implemented. Thanks to the SESAM HAL and an interrupt management unit, the tasks are dynamically or statically executed on heterogeneous computing resources. In addition, multi-application execution is supported by this HAL. A set of scheduling and allocating services in hardware or software can be easily integrated, modified and mixed. Besides, a complete hardware real-time operating system is available, named Operating System accelerator on Chip (OSoC). The OSoC supports dynamic and parallel migration, as well as preemption of tasks on multiple heterogeneous resources, under real-time and energy consumption constraints.

G. NoC

Many NoC topologies are supported by SESAM: a multibus, a mesh, a torus, a multistage and a ring network. These networks are detailed in [9]. All are modeled in approximate-timed TLM. Data exchanges are non-blocking and deterministic, regardless of the network load or the execution constraints. The multibus can connect all masters to all slaves, but does not allow master to master communications. In the mesh or the torus network, one master and several slaves are linked with a router. An XY routing and a wormhole technique are implemented. The multistage is an indirect fully connected network. It is divided into different stages composed of 4 input-output routers, and linked with a butterfly topology. All masters are in one side and all slaves are on the other side. It uses also a wormhole technique to transfer packets. Finally, in a ring network, a message has to cross each router when it goes through a ring. A parameter can change the number of rings. But, each master can connect itself to only one ring. A ring is bi-directional. Besides, we use a fifo with each memory to store memory accesses from computing resources. In order to accept simultaneous requests, two arbiters can be used: a FIFO or a fair round-robin policy. All communications are done at the transactional level and we can accurately estimate

the time spent in every communication.

SESAM offers the possibility to model many different asymmetric MPSoCs with various execution models. It allows the exploration of different approaches for a given set of applications, and the design of new architecture paradigms. Moreover, it helps the sizing of architectures and highlights many important features that cannot be evaluated before the end of the design, such as the control overhead or the effective performance. With its automatic exploration environment and its fast execution (up to 4 MIPS), SESAM brings new facilities to architects.

V. Results

To demonstrate the SESAM's capabilities to model new asymmetric multiprocessors, we have used this framework to carry out the SCMP architecture. The SCMP architecture is a computation-intensive MPSoC that is seen by the CPU as a coprocessor (Figure 5). This architecture is characterized by the centralized hardware operating-system named OSoC, that dynamically executes tasks on heterogeneous processing elements. The SCMP architecture supports a constrained-task and a dataflow execution model, through a logically shared and physically distributed memory. The whole SCMP architecture exists at the Register Transfer Level (RTL) in VHDL. Thus, we had the possibility to add into SESAM, all latencies and constraints, characterized by the Synopsys Design Compiler tool with a low-leakage 0.13 μm@1.2V technology.

A. SCMP modeling

As depicted in Figure 5, the architecture owns three internal NoCs. The system NoC interconnects the external CPU, the external memories and the TLB dedicated to the application, with the core of the architecture. The CPU represents a host interface that allows the user to send on-line new commands to the MPSoC. For instance, it is possible to ask for the execution of new applications. The *TLB Appli* is used to store all the pointers of each task for each application in the external instruction memory. When the simulator starts, it automatically loads all the selected applications into this memory and update the *TLB Appli*.

The control NoC is used to connect the Control Manager and all processing resources through a control interface. In addition, processing resources can communicate with each other, and with the Memory Configuration and Management Unit. The MCMU aggregates the MMU and the CLU presented before. The data NoC is only used for communication between the processing resources and the local memories. It is a multi-bus network that connects all PEs and I/O controllers to all shared and banked memory resources. A specific memory, named *system memory*, is accessible through this network and owns all the system code used by the computing resources.

The OSoC prefetches tasks' code before its execution and manages all the dependencies between tasks. It determines the list of eligible tasks to be executed, based on control and data dependencies. It also manages exclusive accesses to shared resources, and non-deterministic processes. Then, task allocation follows online global scheduling, which selects real-time tasks according to their dynamic priority, and minimizes overall execution time for non real-time tasks. The OSoC is parameterized to support 32 simultaneous active tasks and 8 processors. The time between two successive schedulings, named time-tick, is about 19 μs.

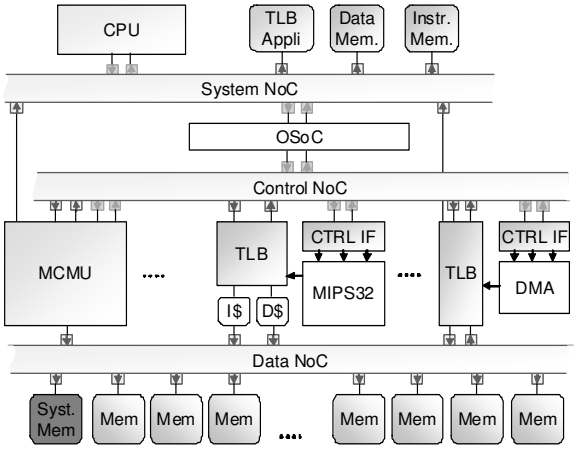


Figure 5. SCMP infrastructure

We have deliberately used a homogeneous architecture composed of Mips32 processors to highlight our parallelism management approach rather than overall system performance. Each processor has a 1KB data and instruction cache exclusive memory and has an access to 64 memory banks of 16KB each. Nonetheless, we use a DMA unit to carry out input image transfers between the internal local memories and the external data memory. ISSs boot on a read-only memory, named *system memory*, that contains all the system code. When the initialization is done, they wait for the Control Manager requests.

The SCMP programming within the SESAM environment necessitates the integration of a Mips32 cross-compiler and the OSoC toolchain. Only the C language is supported for computation tasks. The OSoC needs a Control Data Flow Graph (CDFG) that represents all control and data dependencies of the parallelized application, described with an assembly language that can easily represent CDFGs. Each transition represents an execution constraint that imposes the task execution order. Finally, a parser generates the binary for the control manager from the CDFG file. Besides, the complete system code needed by the platform is decribed by the Mips32 assembly language.

B. Implemented application

SCMP architecture offers a very high degree of parallelism. Thanks to dedicated hardware scheduling and fast reactivity, it also enhances resource utilization. To measure SCMP's performance for dynamic embedded applications, we implemented the connected component labeling algorithm (figure 6). It is a critical application for embedded vision systems

and is particularly relevant to this study in terms of dynamism, parallelism and control dependencies. The labeling algorithm transforms a binary image into a symbolic image so that each connected component is uniquely labeled based on a given heuristic. Connected component labeling is used in computer vision to detect binary regions. Various algorithms have been proposed ([14], [15]) but we have chosen a contour tracing technique that is interesting to stress the architecture [16].

C. Modeling results

We simulated all the platform parameters that affect our results and the OSoC penalties (e.g. communication, control, access time to memories, etc.). We parallelized the initial algorithm by creating independent tasks and carried out the corresponding application graph. We cut the image into sub-images and applied the algorithm on each sub-image. Then, we successively carried out a vertical and a horizontal fusion of labels to analyze frontiers between sub-images. We constructed corresponding tables between labels and changed all labels in sub-images in parallel. To get multiple independent tasks, we executed the application on a 128x128 image, cut into 128 sub-images. At the end, the parallelization brought a new software complexity but involves independent and parallel tasks without modifying the algorithm. An example of extracted results is shown in Figure 7.

As shown in Figure 7-a, thanks to the multi-bank memory architecture and the hardware scheduler, SESAM simulations show that the overhead, due to data accesses (waiting for ready data) and the OSoC scheduling, is rather constant whatever the PE parallelism is. This overhead represents only 3.5 % with 1 PE up to 30 % with 8 PEs of the total execution time. The control is an irregular processing that can only be partially parallelized. This leads to an acceleration factor that can reach around 5 on 8 PEs (Figure 7-b). SESAM demonstrates that a fast migration mechanism can ensure a good occupation of multiple homogeneous resources. These simulations also show how the control overhead is important to the overall performance, and that its optimization can have a very strong impact on the energy and transistor efficiency. Figure 7-c compares the execution length with a multibus and a ring network, while varying the number of computing resources. SESAM shows that the multibus network offers better performances to the architecture. In addition, we underline the limit of the ring network in such architectures. Indeed, between 8 and 12 PEs, the execution time increases by 2 % due to the latency. Finally, Figure 7-d shows the total data cache miss rate in function of its size and the writing policy. SESAM demonstrates that, for this application, a write-back policy with a 512 byte data cache memory is the best trade-off.

SESAM brings the possibility to model a complete asymmetric multiprocessor and to get many useful information on the behavior of the architecture. Most of these information are not obvious and only a real implementation, with accurate timed simulation, can demonstrate the performance of an architecture. For instance, overheads due to data dependencies was difficult to determine before these simulations, as well as the effective acceleration that could be reached with several processors. The simulation speed reaches around 0.5 MIPS for the whole platform, and each execution instance of the application on 128x128 images takes 11 minutes on an Intel Core 2 Extreme X6800 at 2.93 GHz.

VI. Conclusion

This paper presented an asymmetric MPSoC simulator named SESAM. Asymmetric MPSoCs become the solution to execute dynamic embedded applications, while taking care of the transistor and the energy efficiency. This tool allows the design space exploration and optimization of such architectures, while keeping a high accuracy level (90 %). It can also take part in a complete design flow, thanks to its compatibility with standard co-simulation framework. The study of multiple MPSoC instances is eased by a complete environment and an automatic chart generation.

Its performance and capabilities were demonstrated through the simulation of a complete MPSoC architecture. Important results have been easily obtained and the SESAM capacity to size the architecture according to application needs has been shown. The whole architecture speed under the SESAM framework reaches 0.5 MIPS. We now rely on SESAM to improve current research development and bring more innovative results.

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] 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.

[4] 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, 2008.

[5] 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, 2008.

[6] 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.

[7] B. Foote and R.E. Johnson. Reflective Facilities in Smalltalk-80. In *Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, pages 327–335, 1989.

[8] SuperESCalar simulator. http://www.sesc.sourceforge.net/.

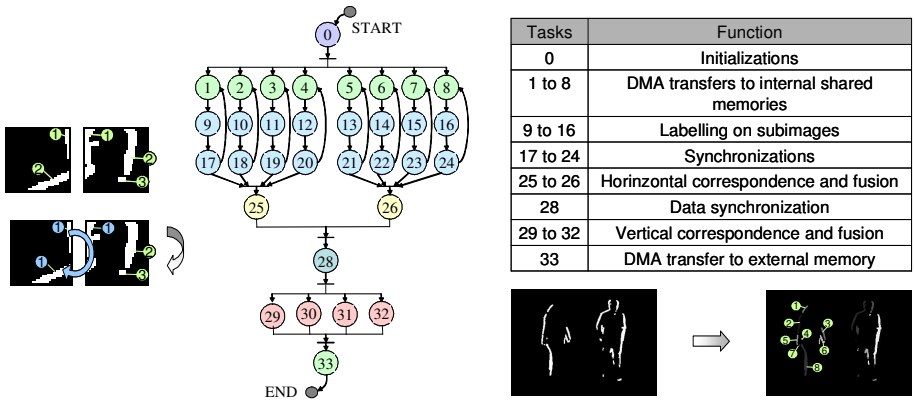
[9] A. Guerre, N. Ventroux, R. David, and A. Merigot. Approximate-Timed Transaction Level Modeling for MPSoC Exploration: a Network-on-Chip Case Study. In *Euromicro Conference on Digital System Design (DSD)*, Patras, Greece, August 2009.

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

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

[12] M. Bartholomeu G. Araujo C. Araujo R. Azevedo, S. Rigo and E. Barros. The ArchC Architecture Description Language and Tools. *Parallel Programming*, 33(5):453–484, 2005.

[13] 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.



Tasks	Function
0	Initializations
1 to 8	DMA transfers to internal shared memories
9 to 16	Labelling on subimages
17 to 24	Synchronizations
25 to 26	Horinzontal correspondence and fusion
28	Data synchronization
29 to 32	Vertical correspondence and fusion
33	DMA transfer to external memory

Figure 6. Labeling application after its parallelization. The labeling is done in all sub-images and two horizontal and vertical fusions are done to get the final labelized image. The corresponding graph is executed by the OSoC, whereas all tasks are processed by MIPS32 ISS.

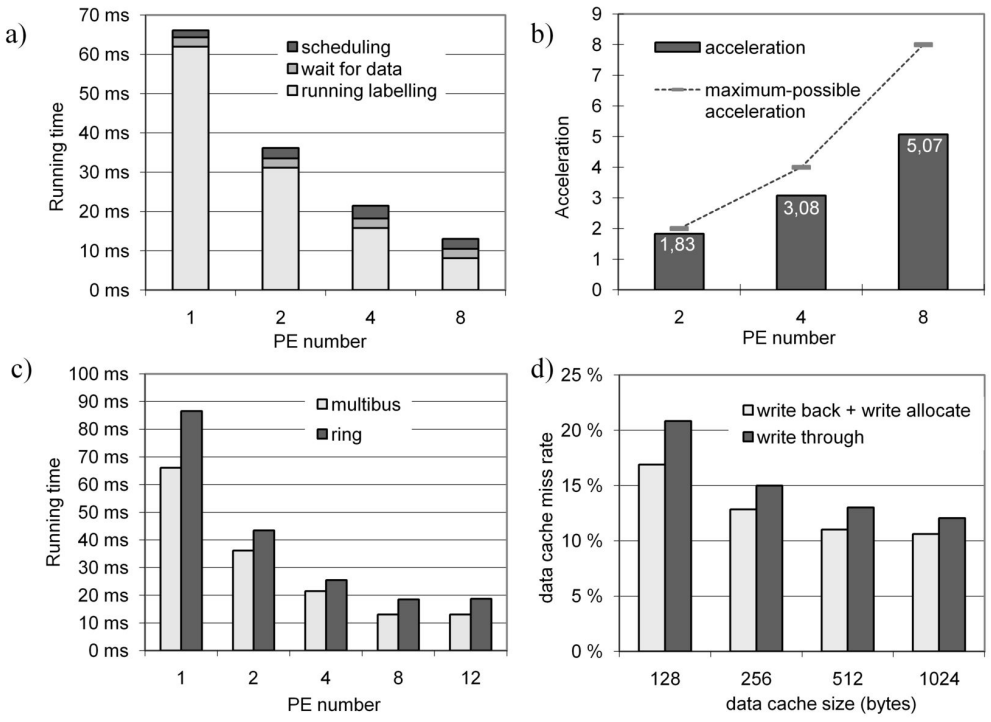


Figure 7. Connected component labeling algorithm execution on the SCMP architecture: (a) total execution time and overhead details depending on the Processing Element (PE) number (Mips32 processors); (b) acceleration rate with multiple PEs versus only one PE, and comparison with the maximum theoretical acceleration that we could obtain without overheads; (c) total execution time depending on the number of PE and two different network topologies; and (d) data cache miss rate with different cache size and writing policies.

[14] I. Horiba K. Suzuki and N. Sugie. Linear-time connected-component labeling based on sequential local operations. *Computer Vision and Image Understanding*, 89(1):1–23, 2003.

[15] L. Lacassagne and B. Zavidovique. Light speed labeling: efficient connected component labeling on RISC architectures. *Journal of Real Time Image Processing*, December 2009.

[16] C.J. Chen F. Chang and C.J. Lu. A Linear-Time Component-Labeling Algorithm Using Contour Tracing Technique. *Computer Vision and Image Understanding*, 93(2):206–220, 2004.