4. System Level Synthesis (SLS)

Group Leader: Frédéric Pétrot
(e-mail: Frederic.Petrot@imag.fr)


Research areas
- Methods and tools for computer aided design and analysis of Multiprocessor System-on-Chip.
- Architectures for communication and memory hierarchies
- Low level software for MPSoC

Financed Projects
ANR: SoCLib, HOSPI
Nano2012: Decopus
Minalogic: Sceptre, OpenTLM, Ciloé
MEDEA+: iGlance, LOMOSA, SoftSoc
Catene: ComCAS, 3DIM3
IST SPRINT, SHAPES, EURETILE
BQR INPG: Damodes, DMSoCProfile

Industrial Partners
STMicroelectronics,
Thales Communications,
Schneider Electric
ATMEL Roma
Target Compiler Technology
Magillem Design Services

4.1 Main trend for embedded systems design: Multiprocessor SoC

We define an embedded computing system as an application specific electronic subsystem used in a larger device such as an appliance, an instrument or a vehicle. The embedded system functions of digital electronics are usually realized using both software running on CPUs and specialized hardware accelerators. The evolution of technologies is enabling the integration of complex hardware platforms in a single chip: called System-on-Chip, SoC. Modern SoC may include one or several CPU subsystems to execute software and sophisticated interconnect in addition to specific hardware accelerators.

Mastering the design of these high programmable and parallel embedded systems is a technical and scientific challenge. It requires new design methods, new design tools, new system modeling strategies to allow for concurrent hardware and software design.

100% of new ASICs include several CPU in 65nm technology. Mobile and Multimedia platforms are multi-processor system on chip (SoC) using different kinds of programmable processors (e.g. DSPs and microcontrollers). Heterogeneous cores are exploited to meet the tight performance and cost constraints. Tomorrow's SoCs are composed of multiple, possibly highly parallel, processors for applications such as mobile terminals, set top boxes, gaming consoles, graphic cards, and network processors. Moreover, these chips contain sophisticated communication networks-on-chips (NoC) to sustain the ever increasing bandwidth requirements. So mastering a huge task level parallelism is the next SoC design challenge. As a result, design methodologies must change their focus to the selection, specialization and usage of processors ---either programmable or dedicated--- as basic components rather than the logic modules. Compared with conventional ASIC design, such a multi-processor SoC is a fundamental change in chip design.

The SLS group is working on both hardware and software architectures for MPSoC systems, and therefore develops CAD tools and methods. This includes low level software to efficiently support the applications, and eases the reuse of legacy code on these complex architectures. Regarding architectures we take benefit from the integration, both because technological evolution, such as 3D integration, or because the number of transistors becomes huge (1.6M Transistors per 1mm² in 45 nm technology).
4.2 Methods and tools for computer aided design


The CAD activities mainly cover the following themes: Modeling, Simulation and Debug and Profiling. Simulation of large scale integrated, potentially heterogeneous, multiprocessor platforms is a long lasting theme of the SLS group. This work is of primary necessity as the number of processors in integrated circuits is rising, and therefore the simulation times are increasing constantly. As a result, the execution of the software on Instruction Set Simulators during simulation (making the processor the ultimate hardware/software interface) is not viable anymore. Several innovative approaches have been defined and demonstrated, and the years 2008-2010 have been particularly fruitful on this topic.

4.2.1 Native simulation strategies

For several years, the so called native simulation approach has been seen as the implementation of the low level software functions of an Operating System using a simulator API. This has become specifically popular with the wide dissemination of SystemC, as it provides many API functions similar to the ones of an OS. The idea is thus to model the system as a stack of layers, and each upper layer can rely on the functions provided by the lower level layers. If the abstraction is coarse, then several layers can be merged to provide at a low execution cost the functions (usually realized on top of an operating system or a high level hardware abstraction such as SystemC TLM), if the abstraction is fine grain, then the "real" realization (some OS code for example) is provided, and the hardware can be described more accurately at the cycle level. One of the abstractions that can be made is illustrated on Figure 4.1. The leftmost part of the figure represents an abstract view of a platform, then details the CPU subsystem that contains one or several CPUs but that share the same operating system (SMP) and software layers. Finally, the hardware view of the subsystem is abstracted.

![Figure 4.1 Software stack representation](image)

The modeling approach is mainly useful for simulating systems at different levels of abstraction. Going high into the abstraction can lead to purely functional information. Addresses may even be hidden behind programming level constructs such as array or variable definitions. However, these approaches have the clear drawback that it is impossible to model operations as frequent and as simple as the copy of data from a hardware IP into a software allocated buffer, because the software buffer is allocated into the Unix process in which SystemC runs, whereas the hardware IP addresses are relative to the target platform. This clearly means that the usage of these approaches is intrinsically limited to trivial platforms.

Our main contribution in native simulation has been to define an approach that allows to simulate a multiprocessor system (whose software enforce the use of the HAL API) using the host machine instead of interpreting the instructions. Figure 4.2 illustrates the approach that relies on the use of the memory of the simulator (in the Unix process) as the memory of the simulated system. This allows avoiding remapping techniques that are not able to handle some situations at the price of a strict adherence to an API (that is very often necessary for porting matters but that may not be used in legacy codes). In order to provide estimations of the execution time, an LLVM based framework is being developed to generate "host code" based on the exact "cross-compiled" code to provide additional information at run time. This approach is however not perfect, as it requires to adhere strictly to a HAL, and to modify the target platform IP's.
step was a necessity to understand the problem, but this solution only goes half-way. A new approach has been researched, and the work on that topic is still ongoing.

### 4.2.2 Dynamic Binary Translation (DBT) based simulation strategies

As far as processor simulation is concerned, the use of dynamic recompilation techniques is very promising from a speed perspective. However, these techniques do not currently provide accurate speed and/or power estimates of the code execution, because the approach “forgets” the original code inherently. Also, the so-called “emulators” are currently fully integrated with their environments, and do not provide the level of modularity that is necessary in the SoC domain. Therefore, we have worked on integrating a DBT engine (QEMU) into SystemC. This has required the definition of a lightweight addition of timing (and/or power) information during dynamic translation, at the basic-block level, as the main issue here is the correct synchronization and time advance between the event-driven simulator and the DBT engine. This also leads to the definition of cache modeling strategies that happened to be quite efficient and accurate. The principle of Dynamic Binary Translation is shown in Figure 4.2. Our contribution has been to generate not only functional calls, but also call allowing to first synchronize with the event driven simulator and second to advance the time.

![Figure 4.2 Dynamic binary translation principle](image)

In order to further increase the speed of simulation for multimedia applications, we have defined an intermediate representation that takes into account SIMD extension, in order to be able to execute target SIMD instructions while benefiting from host SIMD instructions. Using this simple yet innovative approach, we have shown a clear speedup on synthetic benchmarks.

### 4.2.3 Debug and Profiling

We started an activity in debugging by looking at the bugs in concurrent programs running on simulation platforms to ensure first non-intrusiveness and second potential access to all the resources. Profiling benefits from the same infrastructure, with the goal of automatically detecting hot spots from the traces, and pointing out the sources of the hot spots. This work relies on a trace generation infrastructure suited for virtual prototypes, that requires minimal modifications of the models for tracing but changes neither the model behavior nor its timing.

Multiprocessor debug shall not be considered like “usual” debug, i.e. launching a debug process per actual process to follow the flow of computation. As there may be hundreds of cooperating processes, the real problem is to identify wrong data sharing, race conditions and inappropriate understanding of the consistency issues. A formal approach to these issues would be ideal, but unfortunately it is clearly not possible to handle actual programs with these techniques. So we choose a more practical approach in which we analyze execution traces.

Concerning race conditions, as the traces are obtained per processor, they are first reassembled in order to produce a trace per process, and then the communication operation (i.e. load/store, barriers, sync, etc) are ordered (without a notion of global time, but still requiring the memories to be synchronized if the accesses occur on different physical memory banks). Using partial orders instead of total orders (as it would be if we would rely on a global clock for all events) allows to relax the comparison between events, and even though the analysis is trace based, a correct execution can still be detected as leading to a potential race condition. The detection algorithm is exponential, but behaves well in practice if the processes synchronize often, as
these synchronizations are indeed limiting the number of events to be considered which are parallel. Figure 4.3 illustrates the intervals that can be built from a trace.

![Figure 4.3 Interleavings of memory accesses, the black dot represent the synchronizations](image)

As far as consistency violation is concerned, we believe that virtual prototyping allows releasing constraints on what can be verified today. Indeed, the consistency check on actual parallel machines relies on the fact that, among others, each write must be done with a different value in order to be able to know which process or processor produced it. NP completeness of these checks can possibly (it is still to be proved) be made polynomial by adding information that the virtual prototype may be able to produce. Up to now, we have been working on sequential consistency, but we plan to extend the approach to other memory consistency models.

Profiling is another issue. The question is “how in this mass of traces can I identify weak points”, such as phases of a parallel program producing a lot of cache misses, hot spots due to concurrent accesses to the same target, and so on. This problem occurs now very often and we have started cooperation with colleagues from the computer science lab of Grenoble to benefit from their research in data mining. The idea is to automatically identify repeated or costly patterns in the traces and then analyze these patterns, which is a much simpler operation, to identify what leads to the specifically occurring problem. We have started experiments on both performance and power consumption profiling with first interesting results. For example, the tool automatically identified a call to a function to lead to high latencies due to the fact that floating point computations were performed by software in a library. To summarize, we apply here existing data-mining strategies to our traces to detect the occurrence of events that occur either very often or very rarely, and then let the designer analyze the identified points himself. We believe that due to the specific structure of the traces, more efficient algorithms than the general purpose ones can be proposed.

This area is supported by the following projects: OpenTLM, SPRINT, LoMoSA, Comcas, HOSPI, iGlance, SoftSoC, Decopus, Damocles, DMSoCProfile.

### 4.3 Low level software


The low level software is the intermediate software layer between the hardware architecture and the software application layer. This low level software, also called Hardware dependent Software (HdS) includes three main parts:

- The Operating System (OS), which manages the sharing of resources. It is responsible for the initialization and management of the application tasks and communication between them. It provides services such as tasks scheduling, context switching, and so on. The relationship with the hardware is tenuous, and mainly through the implementation of drivers.
- The communication layer, which is responsible for managing the I/O operations and more generally the interactions with the hardware components and the other subsystems. This communication layer implements the different communication primitives used for task communication (intra or inter processor).
- The HAL is a thin software layer, which totally depends on the type of processor that executes the software stack, but also depends on the hardware resources interacting with the processor. The HAL performs the processor related accesses on which the device drivers are based to implement the interface for the communication with the device.
To contribute in this field, we have developed a small embedded operating system that has been ported on top of a few typical processors (MIPS, ARM, SPARC, Microblaze, NIOS, x86, …). This OS serves as basic component for a software design flow able to produce binary code for all computing units of a multiprocessor architecture. The design approach is component-based, static, and very low cost. This design approach is the main contribution in the last 2 years. This allows specifically tailoring the OS regarding the services required by an application, which corresponds to the constraints that an integrated system has to meet.

In terms of communication layer, a set of platform specific device drivers has been developed. However, sharing the device drivers development knowledge is complex, and writing drivers is an error prone and a time consuming activity. Therefore, we are currently working on automatic driver generation from high level hardware and software description. From a practical point of view, this would simplify this task much. A device driver interface can be separated into four parts (see Figure 4.4). The driver requires kernel services like memory allocation, and also offers services (e.g., hardware initialization) to the kernel. b) User application sends general commands to the driver using exported driver interface, while c) libraries provide some services to the driver like string manipulation. d) Lastly, the hardware abstraction layer (HAL) accommodates hardware access methods, which are used by the driver.

![Figure 4.4 Device driver interface](image)

One of the most elementary pieces of information about a device and the driver that manages it, is what function the device accomplishes. Different devices carry out different tasks and need information at different semantic levels to work out. There are devices that play sound samples, devices that read and write data on a magnetic disk, and still other devices that display graphics to a video screen, and so on. Thus, because of the various and interrelated sources of information, driver generation is intrinsically complex. A method, called Me3D, is currently under investigation to help in driver generation. Step by step, information is requested from the designers (IP-XACT model of the platform or device, behavior of the driver, …) for the final generation. A set of tools is also available, called ADDAX (http://tima-sls.imag.fr/www/research/addax/addax-setup/). Methods and tools are today related to DNA-OS, but will be extended to other OS.

This axis is supported by the following projects: SHAPES, EURETILE, SoftSoC, SoCLib, Ciloé, and HOSPI.

### 4.4 Future MPSoC Architectures

*Contributing PhD candidates and Post-Docs: P. Guironnet de Massas, K. Hassan, D. Hedde, Q. Meunier, P.H. Horrein, Y. Xu, H. Jaber, S. Foroutan, F. Dubois, M. Bahmani, J. Tan.*

In our view, the future SoC architectures will be massively parallel, and tend to be more homogeneous (more likely based on identical CPU cores and on coarse grain reconfigurable cores) and will be organized around a network on chip. As a consequence, they will tend to be as hard to program as the parallel machines (shared memory or message passing). One key challenge is to provide support to make their programming easier, specially for memory access. Beside the programming model, the efficiency of MPSoC architectures strongly relies on the efficiency of its cores. Therefore, we are also focused on application specific architectures. The last key challenge is to provide efficient and scalable communication structures, such as 3D-NoC, for these highly parallel architectures. These three key challenges will be described in the following paragraphs.
4.4.1 Memories and programming model

Our work on improving and hiding memory access is focused specifically on shared memory multiprocessor machines, as it has been a widely used approach to concurrent programming. It is also quite natural in integrated systems as the memory bandwidth can be very high, thanks to the use of Networks on Chips. This leads to several choices, detailed now.

- **Make the memory hierarchy transparent and efficient**: this means that caches will be used globally, requiring coherency. This also means that the internal memory will be spread over the chip to ensure a high bandwidth. To optimize the handling of data and instructions, we are currently working on hardware managed page migration techniques to put the necessary information closer to the processors that use it.

- **Optimize the accesses to external RAM**: the access patterns are fundamental to benefit from the highest possible bandwidth for external DRAM accesses. The current memory controllers try to optimize the accesses by building packets of data belonging to the same page, but the memory controller has only a local view of the incoming packets. We try to take benefit from some knowledge available at the Network level, usually available as QoS or priority on virtual channels, in the hope that including this high level information will minimize the latencies and maximize bandwidth.

- **Optimize atomic accesses**: The use of spin-locks has been shown theoretically to be a suboptimal solution to the problem of having synchronized parallel execution progress, and to lead to deadlocks. The introduction of the couple linked load/store conditional or compare and swap has been the advocated solution for years (even though the implementation is not that straightforward). Recently, the introduction of the concept of transactional memories is regarded as an other, more general solution, to the handling of atomicity. Supporting this abstraction requires the implementation of a system called TM, often complex, either software or hardware. In order to investigate the applicability of TM in SoCs, as the support of parallel programming paradigms, we have compared several hardware TM systems based on different architecture choices (cache coherence protocol) and on different conflict resolution policies (i.e. actions undertaken when two or more transactions try to access the same pieces of data). Additionally, since existing TM systems do not ensure starvation freedom, we have also proposed an innovative and efficient conflict resolution policy bringing this guarantee when no cache overflow occurs inside a transaction.

- **Support for unusual concurrent programming paradigms**: POSIX Threads and MPI are the two well known representatives of shared memory versus message passing programming paradigms. For some time, the work stealing idea has been introduced. The principle is that each processor executes its own task until it becomes idle, and then steals a fraction of the remaining work on a randomly chosen busy processor. We have evaluated several implementations of adaptive work stealing and architectural choices to enhance the
performances of work-stealing. We have shown that simple architectural support and wise copy of data can provide up to a 20% performance increase compared to a static parallelization.

4.4.2 Application Specific Processors

As previously stated, it is worth developing efficient supports to ease the programmability and to hide memory management inside highly parallel architectures only if cores themselves use efficiently the silicon area. For years ago the market is already lead by this goal through drastic constraints in terms of time to market and design quality, e.g. embedded systems for telecommunication or multimedia, to develop heterogeneous multiprocessor architectures with application specific processors in order to achieve computation and communication performances.

Therefore, we are working on application specific architectures in two different fields: flexible design radio and 3DTV. As an example, by working on a prototype of 3DTV decoder, we have developed a fixed-point version of a free-view point algorithm, i.e. the 3DTV user can choose its own view point on the 3D scene. The developed version was the entry point of a commercial compilation flow that synthesizes the application-specific processor on FPGA.

Additionally, we are currently targeting dynamically reconfigurable hardware (some processors or IP can be reconfigured on the fly to achieve a different task), since this elegant paradigm enables to have an homogeneous architecture for the silicon point of view, while achieving better efficiency than CPU-based architectures. More precisely, we are currently focusing two solutions to abstract and manage the reconfigurable hardware.

The first solution proposes a hardware manager to control reconfiguration and allocation. The hardware manager is seen as a peripheral by the OS. The second solution is dedicated to flexible design radio, where the manager can additionally take advantage of information sent by the protocol layers to improve the scheduling of reconfiguration.

4.4.3 Three-Dimensional Networks-on-Chip (3D-NoCs)

Three-Dimensional Integration (3DI) is becoming an appealing solution for integration of MP²SoCs (Massively Parallel Multi-Processor Systems-on-Chip). 3D-Integration results in a considerable reduction in the length and the number of long global wires which are the dominant factors on delay and power consumption, and allows stacking dies of different technologies (e.g. DRAM, CMOS, MEMS, RF) in a single package. The incorporation of the third dimension in the design of the integrated systems allows the exploitation of three dimensional topologies which result in a major improvement in network on chip (NoC) performance. Supposing a NoC with a complete 3D-Mesh topology, the number of vertical channels is equal to \(2(N - \sqrt[3]{N})\), where \(N\) is the number of network nodes. As generally each channel of a NoC consists of tens and even in some architectures hundreds of physical wire links, such a network with a large number of nodes requires a huge number of physical vertical interconnections. Through-Silicon-Via (TSV) has the potential to offer a great vertical interconnect density and features an extremely small inter-wafer distance. Even though 3D-Integration using TSVs introduces a whole new set of application possibilities, it also introduces new architecture level design issues. The TSV interconnect pitch (mainly due to its pads) imposes a larger area overhead than the corresponding horizontal wires and a TSV consumes all layers in the upper die in addition to the top layer in the lower die. Additionally, fabricating a 3D integrated circuit using TSVs involves several extra and costly manufacturing steps, and each extra step adds a risk for defects, resulting in potential yield reduction. The yield is an exponential function of defect frequency and the number of TSVs, and thus, exponentially decreases when the number of TSVs goes beyond a certain value. Keeping a homogeneous topology (the same size and shape) for all tiers of heterogeneous chips and using a regular 3D network topology (e.g. fully connected cube) is extremely hard and probably often impossible. Delivering clock to each die and dealing with clock synchronization is another critical problem in the design of the 3D integrated circuits. Furthermore, the run-time temperature variations and stresses and the thermal concerns, which are significant issues of 3D integration, will introduce additional uncontrollable, time-varying clock skew.

To tackle these problems of 3D-Integration technology, we categorize our research on 3D-NoC based system architectural design into three areas:

1) In order to support the network topology heterogeneity of tiers and reduce the number of vertical links (TSVs) used by the network, we have focused on vertically partially connected 3D-NoCs in which the usual planar topologies (normally regular and fully connected, e.g. 2D-Meshes) are partially connected together by only some vertical links and each tier could have its own planar topology independent from other tiers. There are two types of routers: 2D-routers (for example with a switch degree of 5, connected to the four coplanar neighbor routers plus the local subsystem) and 3D-routers (for example with a switch degree of maximum 7, connected to the four coplanar neighbor routers,
plus the local subsystem, plus other neighbor routers placed in the upper and/or lower stages). The goal is not to determine some fixed places for 3D-Routers to construct a fixed 3D-Topology, but to give the system designer flexibility in arranging 2D and 3D routers in any order and any combination as he decides. In other words, the aim is to have no hypothesis and no constraint on the location and connection of vertical links: any node of a tier can be connected to any node of the upper or lower stage. In such irregular 3D topologies the main problem is to define the packet routing strategy (i.e. selecting a path in the network between a source and a destination), as the usual simple algorithmic routings are not applicable. Routing strategies have to provide different guarantees, mainly the freedom from blockage. Deadlock, the situation in which two or more packets are each waiting permanently for another to release a shared channel in a circular dependency, is the most critical blockage. While developing a deadlock free routing algorithm in regular topologies is well understood, proposing a distributed deadlock free routing strategy parameterized by the topology and applicable in irregular topologies is a difficult research topic. By proposing a deadlock-free distributed routing algorithm we have provided a general solution to this issue. We have shown that independently of the size of 2D networks and of the number and placement of the vertical links, the proposed routing algorithm is deadlock and livelock free. We have also developed SoCLib-compatible SystemC CABA model of a router using this routing algorithm. Currently in order to generalize the vertically partially connected topologies to be used in general purpose systems (rather than application specific ones) we are proposing a method to algorithmically determine the place of vertical links.

Figure 4.6 An example of vertically partially connected 3D-NoC

2) In order to overcome the problem of clock skew between layers the use of GALS (Globally Asynchronous Locally Synchronous) approaches is advocated by many 3D integrated systems. A GALS system is divided into several physically independent clusters and each cluster is clocked by a different clock signal. The GALS paradigm also enables the implementation of various forms of DPM (Dynamic Power Management) and DVFS (Dynamic Voltage and Frequency Scaling) methods which, because of heat extraction and energy dissipation, are essential in future 3D-SoCs. Since one obvious way to eliminate the problem of clock skew is the utilization of asynchronous logic, a network with a fully asynchronous circuit design is a natural approach to construct GALS architectures. A large number of locally planar synchronous islands can communicate together via a global three-dimensional asynchronous network. Each link of an asynchronous network works with its own speed as fast as possible, as opposed to the synchronous approach in which the slowest element determines the highest operating frequency. Asynchronous circuits automatically adjust their speed of operation according to the actual local environmental conditions and they are not restricted by a fixed clock frequency. In a 3D-NoC this property offers the opportunity of totally exploiting the potentially high bandwidth of TSVs. This fact that in an asynchronous 3D-NoC the nominal average load of a link is much less than its maximum capacity, especially in GALS systems which the flit injection rate (by synchronous cores) is much lower than the network throughput, encouraged us to search for solutions that make a more efficient usage of TSVs. Serializing the data communication of vertical links is an innovative solution for better utilization of these high-speed connections in an asynchronous 3D-NoC, particularly because 3D integrated circuits are strictly TSV limited to ensure an acceptable yield and area overhead. The serialization allows minimizing the number of die-to-die interconnects and simultaneously maximizing the exploitation of the high bandwidth of these vertical connections, hence
addressing the cost-efficiency trade-off of 3D-Integration using TSV technologies. Additionally, reducing the number of TSVs to be exploited allows for hardware redundancy that is often used to improve the yield. As the principal cause of TSV defects is the misalignments, a simple and effective way to add redundancy and improve yield is to use larger pads. Serialization and consequently using fewer TSVs leads to increase the vertical interconnection pitches and so in the same area we can use square pads larger than standard pads. Another efficient example of hardware redundancy is the use of redundant TSVs which can be used to replace defected TSVs. However the serialization has a direct impact on the network performance. If the serialization degrades the vertical throughput to a value much lower than that of horizontal links, these serial vertical links may become bottlenecks for all paths crossing them. As a consequence, the circuit design optimization of the serializer and deserializer plays a key role. It is also very important to properly determine the optimum serialization level. Hence, our primary desire when designing the serializer and deserializer was to optimize the communication throughput. We have patented our (De)Serializer circuits based on self-controlled (de)multiplexors in 2009.

![Diagram of 3D-Integration with TSVs](image)

**Figure 4.7** Inter-Router Vertical Connections using Asynchronous link of n bits divided into p segments of m bits serialized to one bit

3) In a successful manner, 3D-Integration leads to integrate billions of transistors in a single chip containing a complete large MP²SoC. But the integration of many processing cores in a single system aggravates the inter-core communication traffic congestions and memory organization becomes an issue as it has a great impact on the overall performance of the system. In a 3D-MP²SoC, most of the vertical data communications are due to the request/responses between processors and their blocks of (shared) memory (which could be SRAM, DRAM or NVRAM). Therefore accessing points of vertical links on logic dies perform hotspots of the 3D-SoC which may lead to highly congested areas and thus unpredictable latency, particularly when combining with our two other solutions for 3D-NoCs (i.e. vertically partially connected topologies and serialized vertical links). By providing two complementary approaches, we are analyzing the performance of a 3D-MP²SoC due to different location of memory interfaces over logic dies, location and number of vertical links, and number and location of serialized vertical links as well as serialization level. First, by using a fast analytical method we obtain the performance metrics (such as end-to-end packet/message latency, saturation point, mean buffer utilization, router delay components and the probability of contention, etc) due to different architectural parameters. After the fast exploration of the design space (by the help of our analytical technique) and choosing the most practical combinations (of memory organization and architectural parameters), we use the SoCLib (SystemC) CABA simulation environment to evaluate the performance of selected solutions in a more precise manner.

This area is supported by the following projects: SOCLib, Sceptre, Ciloé, iGlance, 3DIM3
4.5 PhD topics

1. Bahmani Maryam, "Architectural Exploration of 3D-NoCs"
2. Chagoya-Garzon Alexandre, "Abstraction methods for multiprocessor code generation"
3. Chen Hui, “Models and code generation approaches for HW/SW IP integration”
4. Dubois Florentine, “A compositional approach based on analytical formulations for energy and power analysis of NoC architectures”
5. ElMrabti Amin, “Abstract modeling of architectures and applications for system generation”
7. Gligor Marius, “Programming for low power on MPSoC platforms”
8. Guérin Xavier, “Kernels for heterogeneous MPSoC architectures”
9. Guironnet de Massas Pierre, “Transparent Integration of memory hierarchies in MPSoC architectures”
12. Hedde Damien, “Debug of concurrent programs using non intrusive simulation techniques”
15. Lagraa Sofiane, “SoC profiling based on data-mining tech”
16. Meunier Quentin, “HW support for Parallel programming : work stealing and transactional memories”
17. Prost-Boucle Adrien, "Automatic Generation of hardware accelerators on reconfigurable target using high-level synthesis”
19. Shen Hao, “Models for the estimation and analysis of performances of SoC architectures using flexible processors”
20. Tan Jun-Yan, “NoC prototyping”
21. Xu Yan, “Light software services for dynamical reconfiguration”