System Level Synthesis (SLS)

Permanent Personnel
Liliana ANDRADE (MCFCN UGA, section 27)
Olivier MULLER (MCFCN Grenoble INP, section 27)
Frédéric PETROT[(1)] (PREX Grenoble INP, section 27)
Laurence PIERRE (PR1 UGA, section 27)
Frédéric ROUSSEAU (PR1 UGA, section 27)

(1) Team leader and TIMA vice-director

Temporary personnel		
PhD Students	9	Thomas BAUMELA Louis BONICEL Jean BRUANT Maxime CHRIST Georgios CHRISTODOULIS Breytner Joseph FERNANDEZ MESA Bruno FERRES Giulio MILICI Arthur VIANES
Postdocs, engineers, expert collaborators	3	Engineers : Enzo BRIGNON Arthur VIANES Robin STIEGLITZ
Trainees	8	Matheus ALCÂNTARA SOUZA Nadia BENMOUSSA Enzo BRIGNON Amaury BUTAUX Gabriel JOB ANTUNES GRABHER Loic JOVANOVIC Marius LEBLANC Shuo ZHANG
Visitors	1	Manuel SELVA

Research Activities

The SLS team focuses on (a) hardware/software architectures, (b) specification, modeling, simulation and verification of embedded systems on chip and (c) reconfigurable and prototyping for cloud and ad-hoc digital circuits. The work of the team is included in the Laboratory themes “Hardware/software codesign” and “Simulation and verification of systems” described below.

Hardware/Software Codesign

Although the integration techniques have evolved drastically in the last decade, the definition of the frontier between peripheral devices and the kernel of the operating system, structuring how input/output operations occur at runtime, has remained mostly unchanged since the very first days of computers, even though the number of IPs and IPs in SoCs have risen tremendously, has had the number of registers per IP. We have worked on two paths around this topic. The first one deals with generating drivers from abstract descriptions. In practice, it appears to be somewhat contrived, and makes the handling of timing constraints cumbersome and hardly independent of the host OS. So we decided to try a second approach which consists of proposing a generic interface to devices, using channels instead of registers, taking inspiration from USB and virtualization strategies. This strategy has the benefit of partitioning the driver in a front-end, class and operating system dependent, but thus very limited in number, and a backend that is the responsibility of the device maker. We did a first experiment focusing on FPGA IPs, as they are particularly sensitive to version changes. We plan to apply this approach also to hard IPs, taking benefit from the fact that most IPs embed a processor or micro-controller of some sort that can run the backend part of the driver.

Providing high-performance synchronization mechanisms is a key issue to leverage hardware parallelism offered by MPSoCs. We studied the synchronization barrier mechanism and the impact of hardware contention for shared memory clustered MPSoC. Based on the implementation of the TSAR platform on the Veloce2 Quattro emulator, we have designed a non-intrusive approach to observe certain internal signals. We implemented a spy module in the platform to extract at runtime, by this side channel, useful signals like processors program counters and registers. Thanks to that, we have identified hardware modules restrictions and Linux kernel sub-optimal services both in active and passive wait processes. We show how the introduction of delays in the thread awakening process improves the overall synchronization mechanism. We have provided a combined HW/SW optimization. For the passive wait, our proposal provides a 60 % gain for 64 threads running on a 64-core architecture, and about 85 % gain for active wait on 16 and 32-core architecture.

Another work concerns hardware support for synchronization locks. We have confirmed a fundamental hypothesis for the optimization of the lock mechanism: although during the run time a lock may be used by various cores belonging to different clusters, it is often reused by the last core which has released it. Based on this observation, we propose an innovative decentralized solution to manage dynamic re-homing of locks in memory close to the last access-granted core, thus reducing overall access latency and network traffic in case of reuse of the lock by the same cluster.

Field-Programmable Gate Arrays (FPGAs) have been gaining popularity as hardware accelerators in heterogeneous architectures thanks to their high performance and low energy consumption. This argument has been supported by the recent integration of FPGA devices in cloud services and data centers. The potential offered by the reconfigurable architectures can still be optimized by treating FPGAs as virtualizable resources and offering them multitasking capability. The idea to preempt a hardware task on an FPGA with the objective of context switching it has been in research for many years. We nevertheless proposed a new strategy based on flow control analysis to extract the context of a running task from the FPGA to provide the possibility of its resumption at a later time. This strategy minimizes the context size and has the great interest of being applicable to IP generated using high-level synthesis.

Heterogeneity in High Performance Computing HPC nodes appears as a promising solution to improve the execution of a wide range of scientific applications, regarding both performance and energy consumption. Unlike CPUs and GPUs, FPGAs can be configured to fit the application needs, making them an appealing target to extend traditional heterogeneous HPC architectures. However, exploiting them requires an in-depth knowledge of low-level hardware and high expertise on vendor-provided tools, which should not be the primary concern of HPC application programmers. We proposed a framework, which requires the minimum knowledge of the underlying architecture, as well as fewer changes to the existing code. To fulfill these requirements, we extend the StarPU task programming library that initially targets heterogeneous architectures to also support FPGA.

Embedded vision systems are typical of high performance and energy efficiency requirement. As circuit integration is now extremely high, allowing to use hundreds to thousands hardware units in a single die, video stream data movement and parallelisation schemes are the higher level enablers. We focused on a methodology to speed up the design space exploration by rapidly estimating the performance of video analytics application on the STx-ASMP multiprocessor system. This led to tools: Parana rapidly estimates the performance of an application through an analytical approach taking as input application-independent metrics of the platform and some timings of the sequential version of the application. The estimates accuracy are below 5% of a real execution. From the same characterization data, the Tilana tool computes the optimal tile size to parallelize an image processing algorithm.

Artificial intelligence is a very hot topic, but digital hardware acceleration mainly focuses on the use of DSP and ad-hoc MAC acceleration. We have worked on the implementation of fully “unrolled” convolutional networks on FPGA using ternary weights and activations. Thanks to the high amount of resources available in modern FPGA, we have designed and implemented a versatile high-throughput accelerator with high accuracy and low power consumption.

On the general purpose computing side, we have worked on new protocols for ensuring cache coherency in multicore systems. Our main concern is the growing size of the meta-data as a function of the number of cores. A way to limit this is to use multicast, but it is hard to implement on NoCs, or broadcast, but this is highly inefficient in term of throughput and latency. A first approach we worked on consists of introducing virtualization support in NoC so that several virtual NoCs could share a single HW NoC at low HW cost. We were able to demonstrate broadcasting on a subpart of the NoC at lower cost than multicasting. A second approach is to represent the sharing set more efficiently. We propose the use of a rectangular shape to cover most of the sharers, and of a list allocated in a shared heap for the outliers. Using this approach leads to several interesting problems, such as how to find the best covering rectangle, how to implement efficiently the rectangle covering algorithm in hardware, what should be the size of the list and heap, etc.

Simulation and Verification of Systems

The discrepancy between computing and storage motivates the design of memory hierarchies trying to prefetch the data ahead of the computing unit. Both static and dynamic strategies have been explored in the context of image processing. During this period, we focused on dynamic strategies to guess the next data reference through statistical characterization of the past references. The proposed algorithm guesses the next references by online computing of the Kolmogorov-Smirnov hypothesis test and updates some internal parameter thanks to a metric of the prediction quality.

High-level performance estimation of software codes requires abstract models to represent processors behaviors, in terms of instruction throughput and execution latency. To that aim, we have worked on annotating the intermediate representation of a compiler with target specific information while generating code for the host simulation machine. We have discovered that some constructs, loops to be precise, are particularly hard to annotate when aggressive optimization such as loop unrolling, loop folding, loop pipelining are used. Indeed, the mapping between the control flow graph of the target binary and the one of the host binary differ, and therefore the annotation are not accounted for correctly. We worked on host control flow graph reorganization to take into account the machine level optimization and reached a high-level of accuracy at high speed for timing estimation of sequential programs.

Accelerating hardware/software simulation is an historical topic of the group. During the period, we have on the one hand worked on “native simulation” taking into account non-functional properties, applied to many-core platforms and VLIW processors. Even though we obtained interesting and publishable results, we plan not to go further in that direction. Indeed, modeling constraints make native simulation usable only for bare metal software or require very heavy paravirtualization developments. On the other hand, we kept on working on dynamic binary translation, with first a prototype proving that it can be used to execute VLIW codes, and second a strategy which bypasses the logical to physical address translation, that accounts for 10% to 60% of simulation times, depending on the benchmarks. We have mainly focused on speed of simulation, but performance estimation is also important, and to that aim we have modeled micro-architectural properties, such as caches or branch predictors, by annotating generated code. Overall, accuracy in the range 20%/50% can be achieved, but at a high overhead in terms of simulation speed.

Given its importance in modern design, we started an activity aiming at the verification of intrinsic properties of C or C++ embedded software. To that goal, we advocate the use of runtime Assertion-Based Verification (ABV) which aims at checking whether applications obey properties, usually expressed as *formal temporal formulas* that capture the design intent. They can be used for debugging purposes; they can also be useful for the online monitoring of fault-tolerance, security, or performance properties. These properties are automatically transformed into *assertion checkers, triggered upon specific events during execution*. These events can be updatings and readings of program variables (both global and local), and function calls or returns. A specific binary instrumentation mechanism has been defined to implement such an event-driven activation of temporal assertion checkers [CI54].

Highlights

- …
-

Indicators

Scientific production	2019
International journals	2
International conferences	6
National conferences	2
PhD thesis	1

Scientific recognition	2019
Conference/workshop Committees	8

Contracts 2019		
ANR	1	Rakes (2019-2023)
ANRT	4	Kalray (1) / OVH (1) / Schneider Electric (1) / STMicroelectronics (1)
CEC National	1	AI4DI (2019-2022)
EPST	2	Arte (2019-2020) / Digital Hardware AI Architectures (2019-2023)