Virtual Manycore Platforms: Moving Towards 100+ Processor Cores

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

Lieven Eeckhout
Ghent University, leeckhou@elis.ugent.be

Grant Martin
Tensilica Inc, gmartin@tensilica.com

Frank Schirrmeister
Synopsys Inc, fschirr@synopsys.com

Nigel Topham
University of Edinburgh, npt@staffmail.ed.ac.uk

Xiaotao Chen
Huawei Technologies, xtchen@huawei.com

***Abstract*—The evolution to Manycore platforms is real, both in the High-Performance Computing domain and in embedded systems. If we start with ten or more cores, we can see the evolution to many tens of cores and to platforms with 100 or more occurring in the next few years. These platforms are heterogeneous, homogeneous, or a mixture of subsystems of both types, both relatively generic and quite application-specific. They are applied to many different application areas. When we consider the design, verification, software development and debugging requirements for applications on these platforms, the need for virtual platform technologies for Manycore systems grows quickly as the systems evolve. As we move to Manycore, the key issue is simulation speed, and trying to keep pace with the target complexity using host-based simulation is a major challenge. New Instruction Set Simulation technologies, such as compiled, JIT, DBT, sampling, abstract, hybrid and parallel have all emerged in the last few years to match the growth in complexity and requirements. At the same time, we have seen consolidation in the virtual platform industrial sector, leading to some concerns about whether the market can support the required continued development of innovations to give the needed performance. This special session deals with Manycore virtual platforms from several different perspectives, highlighting new research approaches for high speed simulation, tool and IP marketing opportunities, as well as real life virtual platform needs of industrial end users.**

I. Introduction

The fundamental tool for processor based design in today's processor-centric platforms is the Instruction Set Simulator (ISS). As platforms have grown to have more and more processor cores, the performance of ISSs has grown increasingly important in the development of design and verification environments to support designers and users of multi-core platforms. Necessity has been the mother and father of invention – we have seen many techniques arise over the last fifteen years to improve the speed and capabilities of ISSs. Yet technology evolution seems to outpace the evolution of ISSs. As platforms look to have 100+ heterogeneous processor cores, will the ISS technology evolve rapidly enough to allow simulation to keep pace as a design methodology?

In this paper we outline both the requirements that motivate the need for virtual simulation platforms for Manycore platforms and discuss several abstraction and technology developments that give us a chance of being able to meet the performance requirements for these complex design environments in the future.

II. Hybrid Processor Simulation for Virtual Platforms

Instruction Set Simulators (ISSs) for many kinds of embedded processors form the backbone of most virtual platforms. As a consequence, ISS technology has undergone fast evolutions in the past two decades. Early ISSs were based on interpretive simulation, i.e. they emulated every single target instruction by a fixed sequence of host instructions similar to BASIC interpreters. Interpretive ISSs typically achieved a speed of a few KIPS and thus were not of much practical use for complex applications. Compiled simulation [1,2] achieved a speedup of 1 to 2 orders of magnitude by moving large parts of simulation from runtime to simulator generation time. Later, ISSs have been made retargetable by means of architecture description languages like LISA or MIMOLA [3,4]. Another breakthrough was the combination of both ISS approaches and flexibility within the LISATek environment [5], achieving a simulation speed around 10 MIPS (on host machines of that time) for complex target processors. The state-of-the-art today are binary translation based ISSs [6] with a peak performance of >100 MIPS.

Unfortunately, this tremendous progress in ISS technologies is likely to come to a standstill soon. A theoretical speed limit is set by the fact that when finally only a handful of host instructions are needed to simulate a target instruction, there is not much room for improvements. On the other hand, the rapid

978-3-9810801-7-9/DATE11/©2011 EDAA

increase of the number of cores in MPSoC platforms requires much higher simulation speedup than what can be achieved with traditional ISSs.

How to provide ISS technology for 100+ core platforms then? One way to solve this challenge is to utilize parallelism in the simulation hosts (see part IV and [7]). Another way is to increase the simulation abstraction level. While today´s ISSs already support multiple abstraction levels (mostly instruction and cycle accurate), there are use cases which permit sacrificing even more timing accuracy for sake of higher speed. For instance, a SW developer might want to fast-forward a simulation to a certain point of interest (e.g. a breakpoint after the OS boot) and then continue simulation or debugging at instruction accuracy level.

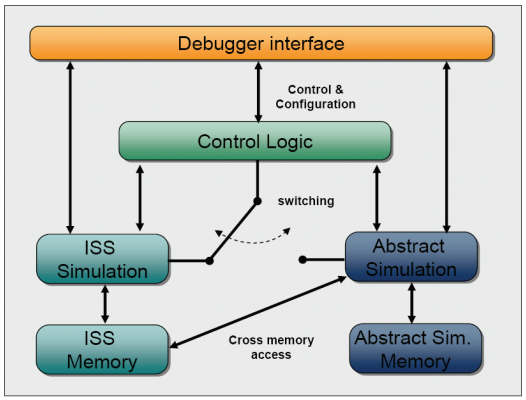

Fig. 1: HySim architecture

HySim [8] provides this functionality by implementing a hybrid between instruction accurate ISS and native (and optionally timing annotated) C code execution on the host. During a simulation/debug session, the user can dynamically switch between "fast" (native code) and "slow" (ISS based) modes. This principle is sketched in fig. 1: A "slow" traditional ISS cooperates with a "fast" abstract simulator (AS) that executes native, yet specially instrumented, C code. The code instrumentation ensures that ISS and AS can be synchronized, e.g. by allowing the AS to access the ISS memory. Obviously, a limitation of HySim is that the source code has to be available, which might not be the case for some legacy or third party code. Furthermore, even in AS mode HySim cannot reach the speed of pure native C code execution due to the need for code instrumentation. However, the speedup of typically 5x-10x versus a binary translation based ISS is significant. Moreover, it has been demonstrated that HySim can be seamlessly integrated into state-of-the-art MPSoC simulation environments like Synopsys Platform Architect and WindRiver Simics. Hence, combined with other speedup technologies, it can contribute to solve the 100+ core simulation challenge.

III. The Virtual Design Requirements of Configurable Manycore Baseband Platforms

The old divisions in systems design between fixed instruction-set architecture (ISA) processors and dedicated hardware blocks have been rent asunder. A growing alternative third way is the use of more and more dedicated application-specific instruction set processors (ASIPs), usually designed with a highly automated flow using a configurable, extensible base ISA and an architectural design language to extend the ISA to application-specific processing and interface requirements [9,10].

The use of ASIPs is growing especially rapidly in the Dataplane, as data and communications-intensive processing requirements become increasingly possible to achieve with configurable processors rather than relying solely on dedicated hardware blocks. As converged devices increasingly combine multiple Dataplane processing functions, such as audio, video, and baseband, the architectures for these devices are increasingly becoming multi-core architecture, combining heterogeneous multi-processing in the Dataplane with single or homogeneous symmetric multi-processing in the control plane.

Dataplane processing for baseband applications are growing in the number of processor cores as the complexity of wireless standards have grown. From 2G through 3G through enhanced 3G we have seen an increase in the number of heterogeneous processors. Long-Term Evolution (LTE) and the future LTE-Advanced increase processing requirements by further orders of magnitude.

Some of the requirements to support single and multi-core simulation for configurable, extensible processors have been covered in [11]. As designers increasingly use more and more heterogeneous cores in the baseband, how do the requirements change? Current LTE PHY baseband solutions for both user equipment and base stations, such as Blue Wonder/Infineon's BWC 200, Docomo's Super 3G, Design Art Networks' 2200 and 2400 and Tensilica's Atlas LTE UE reference architecture [12] use eight to ten heterogeneous Dataplane DSPs as well as control processors and the integrated SOCs they are part of contain other DSPs and control processors for related processing functions.

Since LTE-Advanced will escalate the processing requirements from the LTE level by 5 to 10 times, we can see that the road to Manycore in baseband is firmly established. How are we going to design, verify and debug the complex hardware-software platform architectures developed for this domain? How will we optimize the many heterogeneous processors and mapping of function to architectural components?

There are three areas that need urgent development by the research and commercial tool development community over the next few years:

- Parallel system simulation technologies: offering speed and scale to Manycore systems with hundreds of processors and hundreds of dedicated accelerators and peripherals. Effectively, this must become "the exploitation of multicore to design multicore".
- Debugging concepts and approaches: the current approach of window multiplication for multiprocessors does not scale beyond a few (10 or a dozen) to hundreds. Some new, sophisticated debugging abstractions are needed along with new technology.

- System-level synthesis: With potentially hundreds of ASIPs and many dedicated accelerators, possibly using high-level synthesis to create, we need to reach back into the past and resurrect the idea of true system level synthesis, driven by higher level specifications and able to explore massive design spaces while satisfying complex application constraints.

In fact, research into some of these needs, such as parallel simulation, is already producing some results, as discussed elsewhere in the paper, but there must be a faster and more effective transmission belt between the research community and the commercial ESL tools industry to make the rapid progress designers require.

IV. Ultra-Fast Parallel Simulation in the Manycore Era

The combination of heterogeneous MPSoC architectures with 100+ processor cores, leads to an explosion in the complexity of the design-space. How is a designer to explore this design space to find the most efficient solutions to a given problem? How is the software engineer to develop software in advance of the availability of the next generation of MPSoC devices? How can silicon implementers develop golden reference models of such complex systems and run them in a reasonable time? This requires simulation and virtualization tools that provide architectural observability, ultra-high simulation rates, and cycle-time models of bounded accuracy.

When simulating 100+ cores there is major challenge of scale; the more cores in a system the more time consuming will be the task of simulation. State-of-the-art simulators using just-in-time (JIT) dynamic binary translation (DBT) techniques are able to simulate complex embedded processors at speeds above 500 MIPS. However, these functional ISSs do not provide microarchitectural observability. In contrast, low-level cycle-accurate ISSs are too slow to simulate full-scale applications, forcing developers to revert to FPGA-based simulations. The practical utility of this approach is, however, quite limited. Typically one can achieve emulation rates of around 50 MHz, but accommodate fewer than 16 RISC cores in a single FPGA device. Architectural visibility is also quite restricted in hardware emulators, and there is little scope for experimental evaluation of different alternative designs.

The ARCSIM simulator is able to run ultra-high speed cycle-accurate instruction set simulations at speeds exceeding those of an FPGA [13]. The approach used in ARCSIM can model any microarchitectural configuration, does not rely on prior profiling, instrumentation, or compilation, and works for all binaries targeting a state-of-the-art embedded processor implementing the ARCompact™ instruction set architecture (ISA). In cycle-accurate mode, ARCSIM achieves simulation speeds up to 88 MIPS on a standard x86 desktop computer, whilst the average cycle-count deviation is less than 1.4 % for the industry standard EEMBC and CoreMark benchmark suites. In functional mode, ARCSIM can simulate at rates above 500 MIPS. Functional simulation can be combined with statistical performance models to yield cycle-approximate simulators running at rates between those of purely functional and cycle-accurate models, although with slightly increased estimation error, typically in the region of 3.5% [14].

In simulation there is always a tradeoff between accuracy and speed. One must ask whether the target system state is available for inspection during simulation? Many high-speed simulators do not provide this, but ARCSIM does. This allows the simulator to be used not only as a virtual platform for software development, but also as tool for design-space exploration, and a golden reference model at the center of a hardware-software co-verification environment.

ARCSIM is also a multi-threaded simulator that exploits two distinct forms of parallelism; firstly within the dynamic binary translation process, for higher translation rates and lower translation latency; and secondly for parallel simulation of multiple target processors.

The ability of ARCSIM to perform dynamic binary translation in parallel on a multi-core host yields an additional speedup of up of more than 2x, when simulating a single-core target on a standard quad-core Intel Xeon host system. Its use of an internal LLVM-based JIT compiler into ARCSIM also cuts the overheads of DBT significantly. When simulating a multi-core application, the translations produced for each core may be shared by all cores, as typically happens in SMP systems, or may be private, as often happens in MPSoC systems. Translations may also persist from one simulation to the next, thereby eliminating the translation overhead for common O/S code, for example.

In ARCSIM the transition from single- to Manycore simulation was eased by the modularity of its design, and by its earlier use of multi-threading for parallel translation. Together these provided the essential infrastructure for the move to Manycore simulation in a way that automatically targets multi-core host systems. For example, an ARCSIM simulation will spawn a separate thread for each simulated target processor core, and the host O/S will map these to the available host CPUs. When the dynamic binary translator is periodically invoked, it will dynamically spawn additional translation threads that each translate, compile and link a subset of the “hot regions” of the simulated application. In this way, parallelism of whatever form, is exploited by the underlying host architecture, allowing the simulator to exploit whatever parallel hardware is available at the time.

The key to a practical Manycore virtualization or simulation platform, is to start out with a single core simulation engine that offers ultra-high speed, This must then be coupled with scalability in the multi-threading of multi-core simulations based on that single-core engine.

To put the performance potential of ARCSIM into perspective, consider a 100-core simulation running on a 16-core host. At current ARCSIM performance levels, each cycle-accurate simulated core can expect to be simulated at an average rate of 14 MIPS, equivalent to an aggregate rate of 1,400 MIPS. ARCSIM therefore is therefore poised to deliver the ultra high simulation rates required in the Manycore era.

V. Towards Scalable and Accurate Architectural Simulation of 100+ Core Systems

Architectural simulation is an invaluable tool in a computer architect's toolbox for evaluating design trade-offs and novel research ideas. However, architectural simulation faces two major limitations. First, it is extremely time-consuming: simulating an industry-standard benchmark for a single microprocessor design point easily takes a couple days or weeks to run to completion, even on today's fastest machines and simulators. Second, developing an architectural simulator is tedious, costly and very time-consuming. Architectural simulators typically model the microprocessor in a cycle-accurate way, however, this level of detail is not always appropriate, nor is it called for. For example, early in the design process when the design space is being explored and the high-level microarchitecture is being defined, too much detail only gets in the way. Or, when studying trade-offs in the memory hierarchy, a cache coherence protocol or the interconnection network of a multi-core processor, cycle-accurate core-level simulation may not be needed.

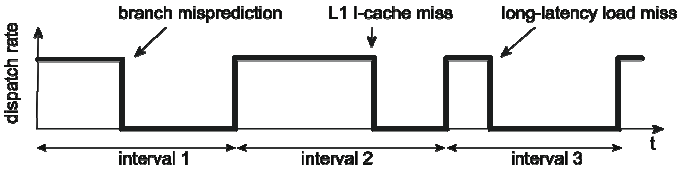

Fig. 2: Interval simulation models timing using intervals delineated by miss events.

Interval simulation [16][17] is a novel simulation approach that aims at raising the level of abstraction in architectural simulation of high-end multicore systems with high-performance superscalar cores. Interval simulation is a fast, accurate and easy-to-implement simulation paradigm; it reduces simulation time by one order of magnitude while at the same time reducing simulator development complexity. Interval simulation raises the level of abstraction in the individual cores compared to detailed simulation: a mechanistic analytical model drives the timing simulation of the individual cores without the detailed tracking of individual instructions through the cores' pipeline stages. The basis for the model is that miss events (branch Mispredictions, cache and TLB misses) divide the smooth streaming of instructions through the pipeline into so-called intervals [15], see also Figure 2. Branch predictor, memory hierarchy, cache coherence and interconnection network simulators determine the miss events; the analytical model derives the timing for each interval. The cooperation between the mechanistic analytical model and the miss event simulators enables the modeling of the tight performance entanglement between co-executing threads on multicore processors.

Implementing interval simulation in two very different infrastructures illustrates its power. The implementation in the University of Michigan M5 simulator [16] shows a one order of magnitude speedup compared to detailed cycle-level simulation with an average prediction error of 4.6% for a set of multi-program and multi-threaded workloads. The implementation in the HP Labs COTSon infrastructure [17] was done in four engineer-months – this illustrates the short development time – and yielded a simulator with a simulation speed in the tens of MIPS range that was validated against real x86 hardware while running complex full-system workloads.

VI. Enabling Productive Manycore Software Development Using Virtual Prototypes

Two of the key issues in Manycore designs are mapping of software to hardware architecture and software debug in the context of hardware. A couple of characteristics make debug especially challenging:

- The type of processing elements used doing the actual computation determines the choice of compilers and debuggers and how specific development tools need to be customized to support specific Manycore architectures.
- The communication on chip and between chips determines how long software tasks have to wait before data can be accessed.
- The types of memory architectures used on and off chip have a profound impact on latencies in the software processes accessing data.
- The structure of the hardware architecture very much impacts whether the programming is straightforward and whether or not it is obvious which portions of a multi-core system are running which tasks.

The characteristics of the actual applications influence the specific types of programming and debug challenges as well:

- Applications can be homogeneous or heterogeneous. Data parallel applications can have multiple channels of audio/video, multiple network streams. Task parallel applications can be pipelined to meet performance (video codec, software radio) or can have a mix of task types, like audio & video. The mapping of application types into Manycore architectures is a non-trivial problem.
- The computational demand of applications determines whether an application can be scheduled at compile time or needs to be scheduled at run time. Totally irregular applications will likely use run-time scheduling. Fixed demands normally imply a static layout and schedule, in which case parallelization of the application and the decision which tasks to run on which processor are made at compile time.
- The way data flows through the system determines its dependency on cache efficiency. When data flow is completely managed by the application using DMA engines, specific communication through FIFOs or local memories then the influence of caches on performance diminishes.
- Generally the programming challenges are most severe when compile time scheduling of tasks is used as opposed to run-time scheduling. When build time configuration is needed for allocation of tasks to processors, automation of optimization becomes very desirable in complex systems. If the apportioning is performed when the application is

decomposed or functions are coded, users need analysis tools to guide their decisions

Overall, when mapping software to Manycore hardware, users are facing both issues of functional correctness and performance. Key debug challenges are

- Data races occur when two or more threads or processors are trying to access the same resource at the same time, where least one of them is changing its state. If the threads or processors are not synchronizing effectively, it is impossible to know which one will access the resource first. This leads to inconsistent results in the running program.
- Stalls happen when users have one thread or processor that has locked a certain resource and then moves on to other work in the program without first releasing the lock. When a second thread or processor tries to access that resource it is forced to wait for a possibly infinite amount of time, causing a stall.
- Deadlocks are similar to stalls, but occur when using a locking hierarchy. If, for example, Thread 1 or Processor 1 locks variable or memory region A and then wants to lock variable or memory region B while Thread 2 or Processor 2 is simultaneously locking variable or memory region B and then trying to lock variable or memory region A, the threads or processors are going to deadlock.
- False sharing is not necessarily an error in the program, but an issue affecting performance. It occurs when two threads or processors are manipulating different data values that lie on the same cache line. On an SMP system, the memory system must ensure cache coherency and will therefore swap the values every time an access occurs.
- Memory corruption occurs when a program writes to an incorrect memory region. This can happen in serial programs but it is even more difficult to find in parallel ones.

Due to simulation speeds, efficiently debugging specific application mappings using RTL simulation is not feasible. Waiting with debug until silicon prototypes are available may lead to issues which cannot be corrected without an expensive re-spins. As a result, pre-silicon and pre-RTL debug techniques become crucially important for Manycore systems.

Virtual platforms offer a solution very early in the project, as soon as the architecture of the design has been worked out. Virtual platforms are a pre-RTL, register accurate and fully functional software model of the System on Chip (SoC), board, I/O and user interfaces. They execute unmodified production code and run close to real-time with external interfaces like USB as ”virtual I/O” directly interfacing to the simulation-machine-native protocol stack. Because they are fundamentally software, virtual platforms provide high system visibility and control including Manycore debug. Since the recent standardization of the OSCI TLM-2.0 transaction-level APIs, SystemC™ has become the suitable infrastructure to develop fast virtual platforms using interoperable transaction-level models and is supported by a variety of commercial products.

VII. A Generic Manycore SoC Virtual Platform

The general trend in processor development has moved from multi-core to Manycore: from dual-, tri-, quad-, hexa-, octo-core chips to ones with tens or even hundreds of cores. A Manycore processor is one in which the number of cores is large enough that traditional multi-processor techniques are no longer efficient. This threshold is somewhere in the range of several tens of cores- and requires a Network on Chip (NoC) to replace bus-based on-chip interconnections. A generic Manycore SoC virtual platform with a configurable NoC is instrumental from architecture exploration, system validation to implementation verification.

One of the configurable NoC virtual platform of choice is derived from NoCBench [18] developed at Texas A&M University. NoCBench is a SystemC based, event-driven simulation environment which has the capability to model and simulate the behaviour of multiple processor cores connected with on-chip network [19]. 2-D mesh is used as the base fabric for NoC with configurable dimensions, routers, arbiters, and other network elements. Each switch in NoC consists of 5 ports, i.e. north, east, south, west ports and 1 central port to the corresponding node. Each node is made of cluster of single-, dual-, or quad- cores plus local distributed memories. The clock frequencies for NoC and cores are also configurable independently. The traffic of NoC consists of packets and flits. The flits are the similar to bus width in bus-based SoC. The transfer of flit is single clock cycle from port to port, port to node if there is no contention. Data flow is packetized into packets, which in turn are sent in flits in multiple cycles based on the mapping of application to nodes and the contention of NoC. The purpose of NoC virtual platform is to get the system latency for each packet, throughput for each port, utilization and contention of NoC. With a Manycore NoC system, it is impossible to get accurate analysis without ESL platform and simulation.

For the early architecture exploration stage before processor cores are finalized, network traffic generators are used to inject different traffic patterns onto the on-chip network. The major patterns range from theoretical to user directed. For theoretical traffics, the common patterns are Point-to-Point, uniform, step, exponential decay, linear decay, truncated exp, truncated linear, react, generic and dynamic. Those patterns are critical to analyze the latency, throughput, utilization and contention of on-chip network.

For user defined application specific data flows, the Intelligent Test Generator (ITG) from Jeda [20] provides a generic interface for NoC simulation. This generic support is based on the following two inputs to decouple from various applications:

- the characteristics of the data flows
- the dependencies among data flows

Besides theoretical statistical analysis of NoC, detailed analysis of all packets from head flit to tail flit is more meaningful for the application specific architecture verification. For NoC simulation, the detailed timing of packet

from node to node, port to port, maximum and average throughput, utilization, and contention are the major benchmarking items for Manycore architecture and systems. In general, head flit to tail flit time of same data flow represents transfer time across NoC, and tail to head time of dependent data flows represents computing time at corresponding nodes. The simulation time is scaleable for 2-D mesh of sizes from 5x5 to 15x15, which are equivalent to 25*M to 225*M cores, where M is the number of cores per node.

These analyses pave the way for core network interfaces and core development on top of NoC.

VIII. CONCLUSIONS

The motivation for Manycore system design has been well-established by the development of more and more advanced communications standards offering greatly increased bandwidth. Portable computing/communications appliances, exemplified by Smartphones, already incorporate 10-20 heterogeneous processing cores and the move to more advanced standards such as LTE Advanced promises to multiply this by another 2-10 times. Heterogeneous ASIPs as a vehicle to design such platforms lead to the need to simulate complex Manycore systems and the road to 100 or more is well laid. Both the portable devices themselves and the infrastructure required to support them such as basestations at macro, micro, pico and femto-levels are good examples of these trends.

Complex applications are mapped to these heterogeneous Manycore platforms, and the need to use simulation is still the primary way to verify application partitioning and mapping, detailed software development, and to debug issues as they arise.

As outlined in this paper, there are several techniques that hold promise for providing a design, verification and debugging vehicle for 100+ Manycore processor platforms. These include hybrid simulation capabilities combining host-based and ISS based simulation; abstractions such as interval simulation; and exploiting parallel processing platforms with advanced techniques for dynamic binary translation and multi-threading. All of these techniques can be exploited to build virtual platforms for the support of design, verification and debug.

We must not forget the need to design the interconnect as well as configure the processing cores in a Manycore platform. New interconnect design methods – networks on chips (NOCs) – themselves can benefit from their own abstractions and statistically-based analysis methods. Separating concerns between the cores and the interconnect wherever possible leads to a more effective and rapidly converging design process.

We must also not forget the need for new debugging approaches that move design teams beyond the need to manage hundreds of debug windows and detailed state descriptions. New abstractions for debug will require considerable research and development in the future.

Combining all these techniques together and developing new ones will be important to ensuring that platform simulation and design performance will keep pace with the rapid development of the technology.

REFERENCES

[1] C. Mills, S.C. Ahalt, J. Fowler: “Compiled Instruction Set Simulation”, Software-Practice and Experience, vol. 21, no. 8, 1991

[2] V. Zivojnovic, S. Tjiang, H. Meyr: “Compiled Simulation of Programmable DSP Architectures”, Journals of VLSI Signal Processing, vol 16, no. 1, 1997

[3] S. Pees, V. Zivojnovic, A. Hoffmann, H. Meyr: „Retargetable Timed Instruction Set Simulation of Pipelined Processor Architectures”, ICSPAT, 1998

[4] R. Leupers, J. Elste, B. Landwehr: “Generation of Interpretive and Compiled Instruction Set Simulators“, ASP-DAC, 1999

[5] A. Nohl, O. Schliebusch, A. Hoffmann, R. Leupers, H. Meyr: „A Universal Technique for Fast and Flexible Instruction Set Architecture Simulation”, DAC, 2002

[6] D. Jones, N. Topham: “High Speed CPU Simulation using LTU Dynamic Binary Translation”, HiPEAC Conference, 2009

[7] C. Schumacher, R. Leupers, D. Petras, A. Hoffmann: “parSC: Synchronous Parallel SystemC Simulation on Multicore Host Architectures”, CODES+ISSS, 2010

[8] S. Kraemer, L. Gao, J. Weinstock, R. Leupers, G. Ascheid, H. Meyr: “HySim: A Fast Simulation Framework for Embedded Software Development”, CODES+ISSS, 2007

[9] Paolo Ienne, Rainer Leupers (eds)., “Customizable Embedded Processors”, Elsevier, 2007.

[10] Prabhat Mishra, Nikil Dutt (eds.), “Processor Description Languages”, Elsevier, 2008.

[11] Rainer Leupers, Olivier Temam (eds.), “Processor and System-on-Chip Simulation”, Springer, 2010.

[12] Chris Rowen, “Resolving the Grand Paradox: Low Energy and Full Programmability in 4G Mobile Baseband SOCs”, CoolChips XIII, Japan, 2010

[13] I. Böhm, B. Franke, N. Topham: Cycle-Accurate Performance Modelling in an Ultra-Fast Just-In-Time Dynamic Binary Translation Instruction Set Simulator, SAMOS, 2010.

[14] D. Powell and B. Franke: Using Continuous Statistical Machine Learning to Enable High-Speed Performance Prediction in Hybrid Instruction-/Cycle-Accurate Instruction Set Simulators, CODES+ISSS, 2009

[15] S. Eyerman, L. Eeckhout, T. Karkhanis, J.E. Smith: “A Mechanistics Performance Model for Superscalar Out-of-Order Processors”, ACM Transactions on Computer Systems, Vol. 27, No. 2, 2009

[16] D. Genbrugge, S. Eyerman, L. Eeckhout: “Interval Simulation: Raising the Level of Abstraction in Architectural Simulation”, HPCA, 2010

[17] F. Ryckbosch, S. Polfliet, L. Eeckhout: “Fast, Accurate and Validated Full-System Software Simulation of x86 Hardware”, IEEE Micro, Vol. 30, No. 6, 2010

[18] S. K. Mandal, N. Gupta, A. Mandal, J. Malave, J. D. Lee and R. Mahapatra, "NoCBench: A Benchmarking Platform for Network on Chip", In Proceedings of Workshop on Unique Chips and Systems, UCAS 2009

[19] Kun Bian, NoCSim User Manual, Texas A&M intern report, 2010

[20] www.jedatechnologies.net