Simplify: a Framework for Enabling Fast Functional/Behavioral Validation of Multiprocessor Architectures in the Cloud

Gabriel Marchesan Almeida*, Oliver Bellaver Longhi‡, Thomas Bruckschloegl*
Michael Hübner†, Fabiano Hessel‡ and Jürgen Becker*
*Institute of Information Processing Technology
Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany
{gabriel.almeida, thomas.bruckschloegl, becker}@kit.edu
‡Embedded Systems Group (GSE)
Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil
oliver.longhi@acad.pucrs.br, fabiano.hessel@pucrs.br
†Embedded Systems in Information Technology (ESIT)
Ruhr-University Bochum (RUB), 44801 Bochum, Germany
michael.huebner@rub.de

***Abstract*—The design of high-performance Multiprocessor Systems-on-Chip (MPSoCs) has proven to be an attractive challenge in embedded systems design automation. However, the complexity of such designs associated with short time-to-market constraints impose serious limitations on the exploration of different configurations and scenarios on the design space exploration. The use of virtual platforms may decrease the time-to-market of these architectures while providing the means to exploit, debug and verify architectures with different features. In this paper, we present the web-based Simplify framework, an interactive approach for MPSoC exploration using an instruction-accurate Open Virtual Platform (OVP). The framework provides an environment to define both software and hardware properties in an intuitive way, and allows designers to validate the functionality as well as the behavior of the modeled architectures at high-abstraction levels. Based on the simulation reports generated from the framework, designers can perform further design modifications and optimizations, and re-validate the whole system in an efficient way, allowing increased design space exploration. For the evaluation of the proposed approach, a set of benchmark applications extracted from MiBench has been used. They run on five different processors (MIPS32, ARM7, OpenRISC (OR1K), PowerPC32 and MicroBlaze) on both mono and multiprocessor architectures and the experiments show considerable simulation speed-ups to obtain application profiling at instruction-level compared to existing approaches based on tracing.**

***Keywords*-virtual platforms, multiprocessor systems-on-chip, application profiling, cloud simulation**

I. Introduction

Multiprocessor Systems-on-Chip (MPSoCs) are platforms that contain multiple processing elements, each one responsible for executing a set of specific functionalities. They are attractive candidate architectures for multimedia processing as these schemes generally can be partitioned in data-dominated functions, which can be processed in parallel on different processors. Those applications however tend to increase in complexity and often exhibit time-changing workloads which makes mapping decisions sub-optimal in a number of scenarios [1]. These factors push designers towards the development of solutions that enable debugging and verification of the whole system in an acceptable time frame. Several models have been proposed in this direction, specially those that allow the simulation and validation of complex and large architectures at different abstraction levels. SystemC Transaction Level Modeling (TLM) provides a standard to enable fast simulation and easy model interoperability for hardware/software co-design. TLM represents a promising alternative for the description and functional validation of large and complex architectures due to its simplicity and faster execution compared to simulators at cycle-accurate level.

Virtual platforms have recently appeared as a promising alternative that enables earlier development and testing of software, reducing SoC schedules and reducing significantly initial development and maintenance costs of embedded software [2]. In [3] authors have proposed a framework that enables multi-layered simulation, where each component of the architecture can operate at different accuracy levels. As case study, the authors model multiprocessor systems-on-chip architectures at different abstraction levels: TLM, CABA (Cycle-Accurate Bit-Accurate) and RTL (Register-Transfer Level). The simulation time of complex platforms may vary according to the abstraction level used for modeling the different components in the architecture. Simulation is very time-consuming due to the number of details that are considered in the component's description. Thus, a single scenario with e.g. 16 processors connected through a network-on-chip may easily take several days of simulation and it can even become unfeasible for larger scenarios.

This paper introduces Simplify, a web-based framework which is intended to be used for modeling large MPSoCs in a fast way through an user-friendly web interface. Section II describes the existing approach for design and validation of embedded systems architectures while positioning the proposed approach in relation to them. Section III introduces the available features in the framework and describes the MPSoC architecture modeling, application description,

978-0-7695-4979-8/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPSW.2013.108

compilation and model execution. Section IV presents the validation scenarios using different platform configurations and benchmark applications, and also illustrates the simulation speed-ups for application profiling using the proposed extended model compared to existing approaches based on tracing information. Finally, Section V draws conclusions about the benefits on using Simplify framework for fast functional/behavioral validation of heterogeneous MPSoCs and brings new perspectives for future works.

II. Related Works

The design of efficient multiprocessor systems-on-chip architectures is currently a hot topic in research and development. Tool vendors like Cadence and Synopsys offer high-end development platforms for this purpose. The Cadence Virtual System [4] platform enables the development of multiprocessor systems with a SystemC TLM based approach where additional libraries with various models for processors as well as peripherals can be used to build up a system. Synopsys Virtualizer [5] addresses the acceleration of both the development and deployment of architectures by using virtual platforms. The development is based on SystemC and the usage of processor modules from ARM is possible.

SimpleScalar [6] is a well-known tool that can perform both cycle and instruction accurate analyzes to extract performance information from processors. In spite of its value, it was not developed to support multiprocessor architectures and a significant software stack is necessary to enable it. Gem5 [7] is a modular platform that provides the integration of different architectures through an object oriented approach. Similar to SimpleScalar, gem5 is mainly used to analyze the processor and its internal components performance like the pipeline and the hierarchy of caches, enabling the execution of different architecture domains and the performance evaluation of such. Therefore, such micro architecture simulation imposes some limits in time with regard to the simulation of massively parallel multiprocessor architectures, representing a prohibitive overall performance when considering the scalability of such architectures.

OVP^{TM} [8], acronym for Open Virtual Platforms, is an active and open-source project which can perform high-speed simulations through instruction accurate prototypes. There are specific APIs to implement processors, peripherals and memory models. The interoperability of models implemented to the platform is done with another API that links components pins in an easy manner. It supports different processor vendors like ARC, ARM, Xilinx, MIPS, OpenRISC, PowerPC and Renesas and they are all provided with optional toolchains and debuggers. The majority of these models are implemented in C but SystemC wrappers are available to be integrated into third-party TLM 2.0 models. The disadvantage of it concerns the limitation of available features in the free license such as application profiling at function level.

All these tools have the goal to reduce time-to-market through the development of complex systems from a high-level of abstraction without the necessity of creating a real prototype for evaluation of the systems capabilities like performance and computation efficiency. Another very relevant topic is certainly the simulation time which is dramatically reduced with virtual platforms.

Instruction or function profiling of embedded applications is very difficult due to limited access to the systems. Most of the existing profiling tools like Intel © VTune^{TM} [9], AMD CodeAnalyst^{TM} [10] or the inbuilt profiler of Microsoft Visual Studio © are used in personal computers. They allow the use of source code instruction or processor sampling in unix or windows based operation systems but are not suitable in embedded systems where operating system support and source code access is very limited. There are some works [11] [12] that implement hardware accelerators to profile embedded software. Another approach is to use software profiling techniques in virtual platforms, but none of the techniques mentioned are supported in virtual platform environments. Only gprof [13], a very old and basic application profiler is implemented in the commercial Imperas tools. Authors in [14] use a low level instrumentation of the embedded application to get the needed information, but this technique requires further modifications in the embedded application itself.

In this paper an application profiler is described that uses the processor model and the virtual platform to provide an instruction profiling that is neither altering the application nor requiring hardware support. The framework is based on OVP, which has been chosen due to its open-source nature that supplies a favorable field for research and development. Another fact that led us to this decision is the fact that the majority of the components available are implemented with high-level ISSs and not with SystemC or pipelined models, improving the scalability of MPSoCs.

III. The Simplify Framework

The idea for developing the Simplify relies on providing a tool where users can easily model and explore MPSoCs with different characteristics. It allows users to describe and execute embedded benchmark applications, validate their functionality while observing their behaviors, and to have access to reports that can be considered for performing further optimizations in the modeled architecture. The tool is available in the project website at http://simplify.itiv.kit.edu and it is a branch of the framework proposed in [15] [16].

A. General Overview

The framework design is essentially divided into two parts: (i) in the *front-end* part users can model either homogeneous or heterogeneous MPSoC platforms by simply dragging and dropping components through a web-based interface. Different settings for processors, memory and interconnection can also be chosen. Moreover, application modeling and allocation for different processors is also possible at this stage. (ii) in the *back-end* part, the framework generates a TLM-based platform accordingly to the specifications defined in the previous stage. At this point users have the possibility to check whether the generated architecture meets the platform specification or not. Once the verification is done, the process moves towards the compilation and execution of the described applications.

B. Tool Flow

There are five models of processor available in the framework (MIPS32, ARM7, OpenRISC (OR1K), PowerPC32 and MicroBlaze) comprising a complete tool-flow including compilation and debugging support. More processor models will be added into future releases of the framework.

1) *Architecture Modeling:* the user specifies the number of processors to be included in the architecture together with the interconnection means (currently only bus connection is available and the network-on-chip is under development).
2) *Processing Element Configuration:* for each processing element users can specify the memory size and processor type.
3) *Application Description:* Simplify supports the execution of applications written in standard C and users can easily integrate their applications into the framework by simply coping/pasting existing source codes.
4) *Application Compilation and Model Execution:* the framework allows the compilation of applications for different target processors. Applications are compiled by cross-compilers installed on the host server running the web-based Simplify framework. Thus, users have the possibility to download a package (tar.gz) containing application source codes, makefiles and the source codes of the generated architecture in order to execute it locally on their host computers.
5) *Execution Reports:* in the end of simulation users have access to execution reports which contain: $i)$ number of simulated instructions (overall architecture and individually per processor), $ii)$ simulation model speed in MIPS and $iii)$ a report containing applications trace and simulation statistics such as MIPS per processor, total MIPS, number of simulated instructions and simulation time.

The proposed extended model also allows users to have access to application profiling information at instruction-level. The profiling is attached to the virtual platform by implementing a profiling function that can be called by the platform after instruction execution. The profiling function increments a counter per instruction that are stored separately from the platform. To achieve model compatibility the instruction list used to implement the instruction counters are extracted from the processor model sources. In addition to the platform functions the OVP processor models are modified to perform the application profiling as follows. The disassembly function of the processor model is used to get access to the instruction lists available within the processor. The function is modified to return the instruction index of the internal instruction arrays. This index is used by the platform to reference and increment the instruction-specific counter. The platform instruction execution has to be changed to enable a single instruction simulation. After each instruction

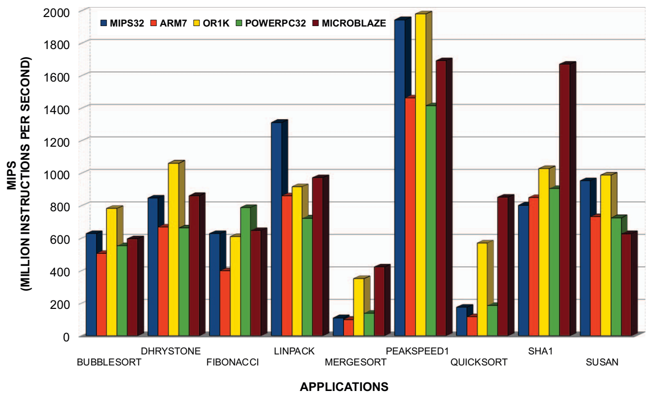

Figure 1. Simulation Performance Evaluation Using Different Types of Processors and Applications

execution the profiling function is called, the index of the instruction retrieved and the instruction counters are updated. If multiple processors are simulated the processor scheduling is modified to simulate one instruction on each processor before starting again with the first processor. After the completeness of the execution, the platform generates an XML file which contains the number of occurrences each instruction of the processor is executed. It is important to observe that most of the state-of-the-art tools allows application profiling at function level and not at instruction-level. Additionally, such profiling tools are intended to run on traditional processors and are not targeted to embedded systems. To the best of our knowledge, our work is the first one to propose a solution for profiling applications with very small granularity of information (instruction-level) using cloud simulation. None of the existing tools previously referred enables model of complex architectures and its validation through cloud simulation.

IV. Case Study

This section introduces a set of scenarios created for evaluating the efficiency of the simulator taking into account different standard benchmark applications, most of them extracted from MiBench [17]. To measure the efficiency of the framework and evaluate the simulation performance, a set containing nine benchmark applications with different processing requirements is used. Figure 1 illustrates the simulation performance in number of MIPS for different benchmark applications running on different processors. It is possible to observe that the simulation performance may vary from processor to processor even running the same application. This is explained by the fact that OVP simulator implements several optimizations based on macro blocks for speeding-up the simulation and its performance may vary according to the model of the used processor as well as the set of executed instructions, which may also vary from application to application.

Table I illustrates the relative performance of the simulator for different benchmark applications and models of

Table I
RELATIVE PERFORMANCE (MIPS) OF THE SIMULATOR USING DIFFERENT TYPES OF PROCESSORS

	MIPS32	ARM7	OR1K	PPC32	MBLAZE
BUBBLESORT	628.4	507.3	784.7	554.2	596.6
	80.08%	64.65%	100.00%	70.63%	76.03%
DHRYSTONE	847.7	669.9	1,062.8	663.2	862.6
	79.76%	63.03%	100.00%	62.40%	81.16%
FIBONACCI	628.9	400.0	610.7	789.3	649.0
	79.68%	50.68%	77.37%	100.00%	82.22%
LINPACK	1,313.4	862.3	918.1	724.0	974.5
	100.00%	65.65%	69.90%	55.12%	74.20%
MERGESORT	110.3	100.0	352.4	139.3	424.0
	26.01%	23.58%	83.11%	32.85%	100.00%
PEAKSPEED1	1,946.0	1,464.5	1,982.3	1,416.5	1,693.9
	98.17%	73.88%	100.00%	71.46%	85.45%
QUICKSORT	475.4	341.8	882.0	444.9	828.8
	53.90%	38.75%	100.00%	50.44%	93.97%
SHA1	804.0	851.6	1,029.2	907.2	1,672.3
	48.08%	50.92%	61.54%	54.25%	100.00%
SUSAN	954.4	732.5	988.6	726.6	627.9
	96.54%	74.09%	100.00%	73.50%	63.51%

processors. The most significative difference is observed when using Mergesort application. The simulation performance obtained when running the application on MicroBlaze processor is of 424 MIPS while achieving only 100 MIPS on ARM7 processors, representing approximately 25% of obtained performance on MicroBlaze. OpenRISC (OR1K) presented to achieve better simulation performance over the others, having obtained the best performance in 5 of 9 analyzed benchmark applications, followed by MicroBlaze ($2/9$), MIPS32 and PowerPC32 (both with $1/9$), and ARM, which has presented low simulation speed performance for all benchmark applications.

In order to better understand the obtained results, we measure the number of executed instructions per processor and per benchmark application. If we consider the Bubblesort application, the processor which executes the largest number of instructions is the MicroBlaze processor, executing more than 600M instructions. However, it presents the best simulation performance compared to other processors running this benchmark (Table I). Now, considering the PeakSpeed1 application, the largest amount of instructions is executed by OpenRISC processor (more than $17,000$M instructions). Thus, this was the same processor that reached the best simulation performance for this benchmark (Table I). However, we cannot say that the larger the amount of instructions executed by a processor, the better its simulation performance will be. When observing Susan benchmark, the largest amount of executed instructions is given by MicroBlaze processor (more than $350,000$M instructions), however when considering the simulation performance, it reaches only 63% of the performance obtained by OpenRISC processor for the same benchmark application. One possible reason for that amount of instructions executed by the standard MicroBlaze processor is due the fact that there is no hardware divider unit for this model of processor, and therefore Susan benchmark uses several instructions that emulate the divider in software.

To verify the scalability of Simplify framework, the application Dhrystone was executed on each processor that belongs to a homogeneous MPSoC with different size. It is possible to observe from Table II that the proposed solution scales well as the number of cores increases. In this example, for running Dhrystone application in a 64 cores architecture, the simulation takes only $1m12s$ to finish its execution, simulating more than $62G$ instructions.

Table II
SCALABILITY OF OVPSIM INSIDE SIMPLIFY

N CORES	MIPS PER CORE	SIM. TIME (s)	SIM. INSTRUCTIONS
4	210.60	4.62	$3,892,315,516$ ($3.9G$)
8	107.40	9.07	$7,792,631,032$ ($7.8G$)
16	53.60	18.14	$15,585,262,071$ ($15.6G$)
32	27.00	36.04	$31,170,524,129$ ($31.2G$)
64	13.50	72.08	$62,341,048,257$ ($62.4G$)

In order to support profiling of applications, the original platform has been extended with the application profiler attached to the OVP MicroBlaze processor model. The platform uses a single instruction simulation mode to capture the currently executed instruction and to build up an application profiling. Each benchmark of the previous simulations has been executed on a platform instantiating one MicroBlaze processor. To this platform a special function providing the instruction information has been attached to count the executions for each instruction available in the processor model. Table III presents the run-time of the simulations using the profiling strategy based on the extended model proposed in this paper. The profiling mechanism is implemented as an extension to the standard model of the MicroBlaze processor provided by Imperas and available on OVPsim. The simulation time considers both user and system time (represented by the elapsed time column) and it is measured for different benchmark applications running during different time period (simulated time).

It is clear to observe that profiled applications takes more time to be simulated compared to standard ones. However, the profiling mechanism using the proposed extended model makes it possible to profile complex application in an acceptable time frame. In comparison to these results, a tracing simulation run of the Bubblesort benchmark in OVP takes more than 26 hours, whereas profiling using the

Table III
PROFILING OF MICROBLAZE PROCESSOR

	ELAPSED TIME (s)	SIMULATED TIME (s)
BUBBLESORT	763.54	6.76
DHRYSTONE	827.27	7.54
FIBONACCI	606.23	5.56
LINPACK	13, 324.86	120.10
MERGESORT	691.74	6.05
PEAKSPEED1	12, 847.09	120.00
QUICKSORT	5, 305.07	47.95
SHA1	2, 149.46	19.42

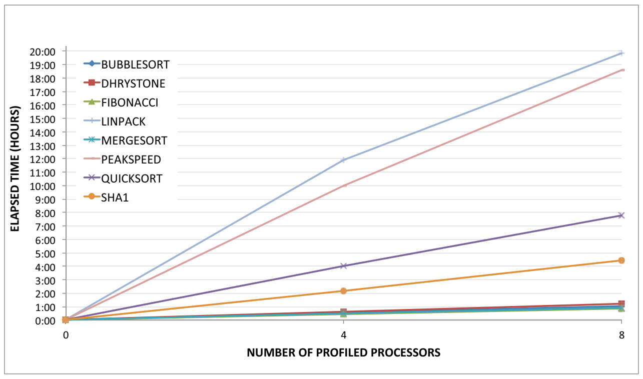

Figure 2. Profiling Run-time of the Extended Model Approach

extended model takes only $12m44s$, reaching a speed-up of around $120\times$. Table III shows that even bigger benchmarks, e.g. Linpack or PeakSpeed1, can be profiled in acceptable time, which would not be feasible using the traditional approach based on tracing information. This is due to two main reasons: (1) when tracing is enabled, all executed instructions must be stored in a file that will be later used as an input for a post-processing application, which will finally count the number of occurrences for each instruction on the target processor.

Considering that the output trace of each executed instruction has approximately 100 bytes of data and considering a run with $1,000$M instructions, the output file for a single application would be around 1GB of size. In the case of Susan benchmark, which executes more than $350,000$M instructions, in order to be capable of executing the profiling based on post-processing information, it would be necessary to have approximately 350GB free space in disk only to store the tracing information. (2) Additionally, it would be necessary to consider the time the post-processing application would take to profile such a big file added to the simulation time for executing an application and writing the profiled instructions into a file. These two main reasons summarize the main motivation for developing an efficient approach which allows application profiling at instruction-level in an acceptable time frame.

The extended platform is capable of simulating and profiling multiprocessor systems. Within the platform eight MicroBlaze processors have been instantiated to show the scalability of the proposed profiling approach to multiprocessor platform simulations. Table IV illustrates the results of simulation run-time and simulation performance with different profiling configurations. These configurations differ in the amount of profiled processors from zero to eight. In most cases it is sufficient to profile only one desired processor in a multiprocessor architecture. Therefore non-profiled processors run an increased interval of simulated instructions to speed up the simulation. The results in Table IV show that it is feasible to profile a single core or up to four cores in a multiprocessor platform. Even profiling up to all eight cores is acceptable depending on the application. The most critical benchmark (LinPack) is capable to be profiled in less than 20 hours in all eight cores. It is really important to notice that the approach based on tracing information would not be feasible at all, taking several weeks of simulation and being limited according to the disk space needed for storing the whole application tracing to be used for post-processing. The results of the simulations presented in Figure 2 show that the simulation time follows a linear approximation of the elapsed simulation time proportional to the number of profiled processors per benchmark. This makes simulation time easily predictable for a different number of cores to be profiled.

V. CONCLUSION

Virtual platforms are an important approach to reduce time-to-market and design complexity. To design real MPSoCs is mandatory to have application profilers, intuitive architecture builders and simulators in order to make a faster design space exploration. These characteristics can be accomplished adopting OVP as base of development due to its non-monolithical nature and its possibility to scale the number of processors without overcharging simulation performance.

We highlight the two main contributions of this work as the application profiler and the intuitive framework to build MPSoCs. The first one is a web-based environment to build, compile and simulate different platforms running several applications, allowing users to better explore the design space. The framework uses a graphical interface with drag and drop components which simplifies the usability of virtual platforms for end-users and allows cloud simulations without the need of having to configure the virtual environment. The second contribution uses a specific approach attached to the platform to get more accurate execution details. The proposed approach allows designers to profile applications at instruction-level and achieves a speed-up of more than $120\times$ compared to the existing approach using simulation tracing. The results show that this approach can also be used for profiling multiprocessor architectures within acceptable time slots. To the best of our knowledge, this is the first

Table IV
PROFILING EFFICIENCY OF THE EXTENDED MODEL APPROACH USING MULTIPROCESSOR ARCHITECTURES

N PROFILED PROCESSORS	0/8		1/8		4/8		8/8	
	MIPS PER CORE	ELAPSED TIME (s)	MIPS PER CORE	ELAPSED TIME (s)	MIPS PER CORE	ELAPSED TIME (s)	MIPS PER CORE	ELAPSED TIME (s)
BUBBLESORT	103.2	6.55	1.3	537.73	0.3	1,949.43	0.2	3,676.84
DHRYSTONE	144.7	5.21	1.1	673.76	0.3	2,302.94	0.2	4,345.39
FIBONACCI	128.2	4.34	1.3	435.64	0.3	1,631.70	0.2	3,074.54
LINPACK	164.1	73.17	1.2	9,776.50	0.3	42,869.31	0.2	71,526.23
MERGESORT	44.8	13.51	1.2	491.92	0.3	1,770.59	0.2	3,263.35
PEAKSPEED1	312	38.46	1.3	9,160.84	0.3	36,025.82	0.2	66,930.06
QUICKSORT	111.9	42.86	1.2	3,885.49	0.3	14,443.82	0.2	28,045.93
SHA1	46.6	41.70	1	2,041.06	0.2	7,875.74	0.1	15,970.25

approach to enable application profiling at instruction-level on virtual platforms achieving good simulation speed.

Future works include: a) NoC support that will allow more flexibility to the designer in terms of interconnection possibilities; b) operating system support for enabling multi-threading execution allowing the assignments of distributed tasks and evaluation of real-time constraints at run-time; and c) advanced profiler support combining instruction and function profiling to allow application optimizations at higher abstraction levels.

ACKNOWLEDGEMENTS

This research was sponsored by the European Commission under the 7th Framework program within the FlexTiles project (FPT ICT-288248).

REFERENCES

[1] G. M. Almeida, G. Sassatelli, and et al, “An adaptive message passing mpsoc framework,” *International Journal of Reconfigurable Computing*, vol. October, 2009.

[2] Imperas-Ltd, “Open virtual platforms (ovp),” http://www.ovpworld.org, 2012.

[3] C. Roth, G. M. Almeida, and et al, “Modular framework for multi-level multi-device mpsoc simulation,” *The 18th Reconfigurable Architectures Workshop (RAW’2011)*, May 2011.

[4] Cadence, “Virtual system platform,” http://www.cadence.com/products/sd/virtual_system/pages/default.aspx, 2012.

[5] Synopsis, “Tools to build, distribute and use virtual prototypes and vdks,” http://www.synopsys.com/Systems/VirtualPrototyping/Pages/Virtualizer.aspx, 2012.

[6] S. LLC, “SimpleScalar Tool Suite,” http://simplescalar.com.

[7] T. gem5 Simulator System, “The gem5 Simulator System - A modular platform for computer system architecture research,” http://gem5.org.

[8] OVP, “Open Virtual Platform,” http://ovpworld.org, 2012.

[9] Intel, “Intel vtune amplifier xe 2013,” http://software.intel.com/en-us/intel-vtune-amplifier-xe, 2013.

[10] AMD, “Amd codeanalyst performance analyzer,” http://developer.amd.com/tools/heterogeneous-computing/amd-codeanalyst-performance-analyzer/, 2013.

[11] L. Shannon and P. Chow, “Using reconfigurability to achieve real-time profiling for hardware/software codesign,” in *in FPGA 04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*. ACM Press, 2004, pp. 190–199.

[12] R. V. Peri, S. Jinturkar, and L. Fajardo, “A novel technique for profiling programs in embedded systems,” in *In ACM Workshop on Feedback-Directed and Dynamic Optimization (FDDO-2)*. ACM Press, 1999.

[13] S. L. Graham, P. B. Kessler, and M. K.Mckusick, “Gprof: A call graph execution profiler,” in *Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, ser. SIGPLAN ’82. New York, NY, USA: ACM, 1982, pp. 120–126. [Online]. Available: http://doi.acm.org/10.1145/800230.806987

[14] T. Carmel-Veilleux, J.-F. Boland, and G. Bois, “A novel low-overhead flexible instrumentation framework for virtual platforms,” in *Rapid System Prototyping (RSP), 2011 22nd IEEE International Symposium on*, may 2011, pp. 92 –98.

[15] A. Aguiar, S. Filho, and et al, “Hellfire: A design framework for critical embedded systems’ applications,” *The International Symposium on Quality Electronic Design (IESQD)*, pp. 730 –737, march 2010.

[16] F. G. Magalhaes, O. Longhi, and et al, “Noc-based platform for embedded software design: An extension of the hellfire framework,” *The International Symposium on Quality Electronic Design (IESQD)*, march 2012.

[17] M. R. Guthaus, J. S. Ringenberg, and et al, “Mibench: A free, commercially representative embedded benchmark suite,” in *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*, ser. WWC ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 3–14. [Online]. Available: http://dx.doi.org/10.1109/WWC.2001.15