#HiPEAC18

January 22-24, 2018, Manchester, United Kingdom

HIPEAC 2018 Conference

Proceedings of

RAPDIO 2018 Workshop

Manchester, United Kingdom

22th January 2018

#HiPEAC18

January 22-24, 2018, Manchester, United Kingdom

Organizing committee

Daniel Chillet, *University of Rennes 1*
Reda Nouacer, *CEA List*
Morteza Biglari-Abhari, *University of Auckland*
Daniel Gracia Pérez, *Thales Research & Technology*
Gianluca Palermo, *Politecnico di Milano*

Program committee

Mario Porrmann, *Bielefeld University*
Roberto Giorgi, *University of Siena*
Philipp A. Hartmann, *Intel*
Jeronimo Castrillon, *TU Dresden*
Sotirios Xydis, *National Technical University of Athens*
Michael Huebner, *Ruhr-University Bochum*
Tim Kogel, *Synopsys*
Frédéric Pétrot, *TIMA Lab, Grenoble Institute of Technology*
Antonino Tumeo, *Politecnico di Milano*
Pierre Boulet, *Univ Lille 1, CRIStAL*
Davide Quaglia, *University of Verona*
Christian Haubelt, *University of Rostock*
Alper Sen, *Bogazici University*

Website

`http://www.rapido.deib.polimi.it`
`http://www.rapido.deib.polimi.it/RapidoProceedings.pdf`

Schedule

— Session 1 — 10 : 00 − 11 : 00

— **Keynote 1 Nikil Dutt,** University of California, Irvine
Self-Awareness for Heterogeneous MPSoCs : A Case Study using Adaptive, Reflective Middleware

— Session 2 — 11 : 15 − 12 : 45

— **Keynote 2 Tim Kogel**, Synopsys
Building Smart SoCs - Using Virtual Prototyping for the Design of SoCs with Artificial Intelligence Accelerators

— Rabab Bouziane, Erven Rohou and Abdoulaye Gamatie
Compile-Time Silent Store Elimination for Energy Efficiency : an Analytic Evaluation for Non-Volatile Cache Memory

— Gereon Onnebrink, Rainer Leupers and Gerd Ascheid
ESL Black Box Power Estimation : Automatic Calibration for IEEE UPF 3.0 Power Models

— Session 3 — 14 : 00 − 15 : 30

— **Keynote 3 Alberto Bosio**, Lirmm, Montpellier, France
Cross-Layer system-level reliability Estimation

— Giovanni Liboni, Julien Deantoni, Antonio Portaluri, Davide Quaglia and Robert De Simone
Beyond Time-Triggered Co-simulation of Cyber-Physical Systems for Performance and Accuracy Improvements

— Ahmet Erdem, Davide Gadioli, Gianluca Palermo and Cristina Silvano
Design Space Pruning and Computational Workload Splitting for Autotuning OpenCL Applications

— Vittoriano Muttillo, Giacomo Valente, Daniele Ciambrone, Vincenzo Stoico and Luigi Pomante
HEPSYCODE-RT : a Real-Time Extension for an ESL HW/SW Co-Design Methodology

— Session 4 — 16 : 00 − 17 : 30

— **Keynote 4 Guy Bois**, Polytechnique Montreal and President of Space Codesign Systems
Specific needs for the modelling and the refinement of CPU and FPGA platforms

— Irune Yarza, Mikel Azkarate-Askasua, Kim Grüttner and Wolfgang Nebel
Real-Time Capable Retargeting of Xilinx MicroBlaze Binaries using QEMU

— Asif Ali Khan, Fazal Hameed and Jeronimo Castrillon
NVMain Extension for Multi-Level Cache Systems

— Alexandre Chabot, Ihsen Alouani, Smail Niar and Reda Nouacer
A Fault Injection Platform for Early-Stage Reliability Assessment

List of regular papers

Work in progress paper

ESL Black Box Power Estimation : Automatic Calibration for IEEE UPF 3.0 Power Models

ESL Black Box Power Estimation: Automatic Calibration for IEEE UPF 3.0 Power Models

Gereon Onnebrink
Institute for Communication Technologies and Embedded Systems
RWTH Aachen University, Germany
onnebrink@ice.rwth-aachen.de

Rainer Leupers
Institute for Communication Technologies and Embedded Systems
RWTH Aachen University, Germany
leupers@ice.rwth-aachen.de

Gerd Ascheid
Institute for Communication Technologies and Embedded Systems
RWTH Aachen University, Germany
ascheid@ice.rwth-aachen.de

ABSTRACT

Power-aware design space exploration at early electronic system level (ESL) is highly facilitated by virtual platforms. In order to define and exchange power models, the IEEE standard 1801-2015 – UPF has been defined to allow developers, vendors and customers seamless usage of the same power models. However, one obstacle is still not covered by the standard: it is not specified and solved how to construct the model in the first place. This paper offers a solution to close this gap. A well proven semi-automatic black box ESL power estimation methodology [17], which allows to have power estimates with approximately 5 % error, is combined with UPF. Although the standard is based on power state machines and requires another tracing method, a procedure is presented to overcome these differences. Representative case studies show that similar estimation accuracy can be achieved.

CCS CONCEPTS

• **Hardware** → Power estimation and optimization; Electronic design automation; • **Computing methodologies** → Modeling methodologies; • **General and reference** → Measurement;

KEYWORDS

Electronic system level, power model, power estimation, digital signal processor

ACM Reference Format:
Gereon Onnebrink, Rainer Leupers, and Gerd Ascheid. 2018. ESL Black Box Power Estimation: Automatic Calibration for IEEE UPF 3.0 Power Models. In *RAPIDO: Rapid Simulation and Performance Evaluation: Methods and Tools, January 22–24, 2018, Manchester, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3180665.3180667

1 INTRODUCTION

The ever increasing computational workloads and tighter constraints, such as power budget, battery lifetime and sufficient performance, have to be tackled by the next generation of Multiprocessor Systems-on-Chip (MPSoCs). To provide the best trade-off between cost, power and performance, these platforms generally incorporate diverse processing elements, e.g., general purpose processors, DSPs and GPUs, along with target application optimised communication network topologies and memories. Short time-to-market cycles bear huge challenges to the Design Space Exploration (DSE) of such complex MPSoCs. While most accurate values for power consumption and performance can only be gathered at late stages of the design flow, design decisions made at Electronic System Level (ESL) have a much stronger impact [25].

Virtual Platforms (VPs) enable hardware and software co-design. The industry standard approach is to use SystemC Transaction Level Modelling (TLM) [3] for DSE at ESL. On the one hand, timing simulation is a stable feature and can be set from cycle accurate over instruction accurate to pure functional simulation to fit the need of the MPSoC designer. On the other hand, power estimation is still under development. Academia has proposed many different approaches how power-aware simulations at ESL can be achieved. Some of them have already been adopted and shipped in industry standard tools. To further unify the community, a common way of building and exchanging power models for all components of an MPSoC is required. Usually, IP vendors want to give such models to their customers or share them internally. The former usually implies that the component model is shipped as binary code, i.e. black box. Hence, less inputs are available to drive a shareable power model. Therefore, the IEEE standard 1801-2015 was extended to provide an interface [2]. In its current version 3.0, the Unified Power Format (UPF) supports SystemC TLM VPs and uses Power State Machines (PSMs) as a reasonable modelling approach for highly accurate power estimates (cf. Section 3.2).

Unfortunately, the standard lacks one important detail. There is no definition or procedure how a power model can be constructed in the first place. In general, an indispensable requirement is the need of a reference in form of spreadsheets, lower level simulations from previous design cycles or even hardware measurements. Besides that, a method of calibrating the UPF power model with this reference is strongly desired. To close this gap, this paper proposes an extension for the well proven ESL black box power estimation methodology, presented in [17]. In order to enable automatic power model calibration for UPF, the following contributions are presented:

- Extending the ESL black box power estimation method to support tracing of UPF relevant states.
- Generating PSM power models out of the ESL power estimation methodology approach automatically.
- Evaluation of the proposed methodology for ARM Cortex-A9 and Blackfin 609 DSP processors.

RAPIDO, January 22–24, 2018, Manchester, United Kingdom

ACM ISBN 978-1-4503-6417-1/18/01…$15.00
https://doi.org/10.1145/3180665.3180667

- Comparison with an industry standard tool that supports UPF power models.

2 RELATED WORK

Over the last decade, intensive research has been conducted in industry and academia to find solutions for power-aware DSE at ESL. Three industry standard tools are the *Virtualizer* tool set from Synopsys [23], *Aceplorer* from Intel Docea [1], and *Vista* from Mentor Graphics [25].

Trying to enable fast and power-aware system level simulations is already done in early work by using equations or table based power models for each single instruction with a cycle accurate architectural simulator [5, 6, 27]. More recent work adopts this technique and enables it for cycle [21] and fast instruction-accurate SystemC simulations [19]. The concept of functional level power analysis (FLPA) is a more abstracted variant and presented in [13]. Using FLPA, a power model is built for every functional unit. Newer work combines FLPA with SystemC simulations and extends it from processor models to the entire MPSoC platform, e.g. models for caches. Measurement on real hardware is performed as calibration reference for each functional unit [12, 16].

Another approach to build power models is to use finite state machines, so called PSMs. Such a PSM can be driven from a functional SystemC simulation [22]. Possible covered states are active and idle power, and dynamic voltage and frequency scaling affected states. The works [7, 9] make use of the TLM time annotation style and the extension mechanism of TLM to implement and drive the PSMs. There is also work that relies on an older version of the IEEE-1801 standard [4]. But the question how to generate reasonable power states is not answered.

Besides power models for processors, investigations are conducted with focus on memory and communication networks of MPSoCs. A power model has been built for a specific 3D-DRAM system in [11]. A data sheet serves as input for the power values and the simulation of the DRAM control commands gives the actual timing behaviour. This procedure is formalised with Petri Nets in [10]. Looking into communication architectures, a framework that deals with bus matrices is introduced in [15] and uses energy macro models to obtain the power and energy consumption. A linear combination of RTL signal traces is linked into SystemC TLM modules to estimate power and energy. Generating power models in a similar way, different communication architectures are investigated in [18]. A semi-automatic curve-fitting approach is used to calibrate the power models from ESL traces and a reference, such as measurements of actual hardware or simulation at lower abstraction levels.

All previously mentioned approaches have in common that the power model has to be obtained in the first place, but except the approach of [18], no generic applicable method is given. Further, insight into the models is necessary to drive the power estimation. To overcome the latter, a black box power estimation methodology is presented and evaluated in [14, 17]. The contributions of these three works are the basis of this work, where a semi-automatic generation of UPF power models is proposed.

$$\mathbf{P}_{\text{est}} = \mathrm{S} \cdot \mathbf{a} = \begin{pmatrix} \mathbf{1} & \mathbf{s}_2 & f \cdot \mathbf{s}_3 \end{pmatrix} \cdot \begin{pmatrix} \mathrm{a}_{\text{fi}1} \\ \mathrm{a}_{\text{fi}2} \\ \mathrm{a}_{\text{fd}3} \end{pmatrix}$$

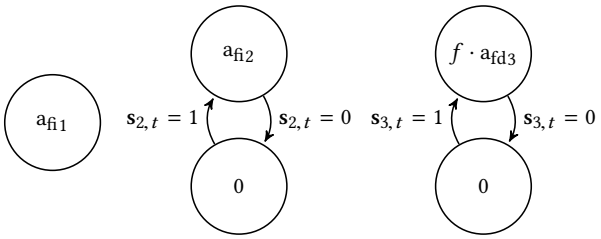


Figure 1: Simplified example of a linear power model and the corresponding PSMs

3 POWER ESTIMATION METHODOLOGY

The ESL power estimation methodology proposed in [17, 18] is the key element for semi-automatic calibration of UPF power models. The concepts are first presented in [18] and modified to support black box power estimation [17]. Both are briefly summarised below. After that, an introduction to UPF is given, together with the approach of generating PSMs out of the linear power model.

3.1 Linear Power Model

The CMOS hardware power model is composed of two parts, the constant leakage power and the dynamic power, which is dependent on short circuit and switching activity. The dynamic actions performed by the CMOS chip are driven by control signals initiated by the application. Thus, tracing these control signals is enough to estimate the dynamic power.

Let there be $N-1$ control signal traces, called ESL traces, each represented by a column vector $\mathbf{s}_i \in \mathbb{R}^T$, recorded over T sampling periods of the length t_{samp}. To model the constant leakage power, the first trace is set to 1, i.e. $\mathbf{s}_1 = \mathbf{1}$. Combining all traces results in the compact representation of the matrix $\mathrm{S} \in \mathbb{R}^{T \times N}$. Assuming a linear relation between the traces and the power consumption, the estimate can be calculated as shown in Equation 1, where $\mathbf{a} \in \mathbb{R}^N$ denotes the so-called *power model factors*.

$$\mathbf{P}_{\text{est}} = \mathrm{S} \cdot \mathbf{a} \tag{1}$$

Originally, this power estimation methodology supports only fixed frequency simulations. But in modern MPSoCs, Dynamic Voltage and Frequency Scaling (DVFS) is a common approach to provide a trade-off between power and performance. With a modification, it is possible to build frequency aware power models by adding the current frequency in Equation 1 [17]. With this, explicit information about the frequency is added to the implicit one stored in each ESL trace.

$$\mathbf{P'}_{\text{est}} = \mathrm{S'} \cdot \mathbf{a'} = \begin{pmatrix} \mathrm{S} & f \cdot \mathrm{S} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{a}_{\text{fi}} \\ \mathbf{a}_{\text{fd}} \end{pmatrix} \tag{2}$$

where $\mathbf{a}_{\text{fi}}, \mathbf{a}_{\text{fd}} \in \mathbb{R}^N$ denote the **f**requency **i**ndependent and **f**requency **d**ependent power model factors, respectively. An example linear power model can be seen in the upper part of Figure 1.

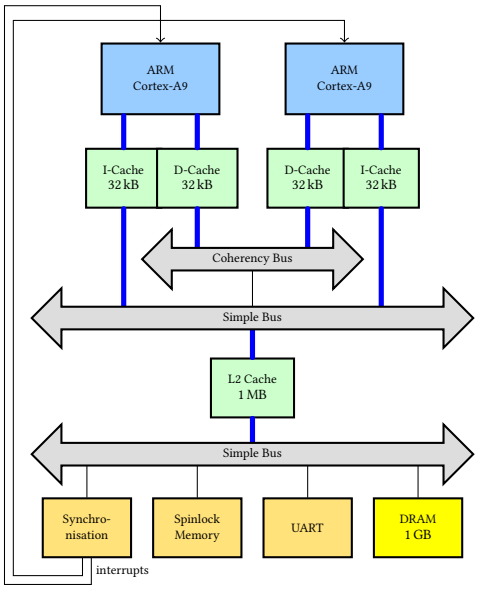


Figure 2: Virtual platform for the OMAP4460 ARM Cortex-A9 subsystem on the PandaBoard

3.2 IEEE 1801-2015 – UPF

The IEEE standard 1801-2015 [2], also known as UPF is the third refinement and supports as new feature ESL power modelling and analysis in virtual prototyping applications. Due to its history of providing an HDL-independent way of specifying power at early stages in the design process, such as register transfer level, UPF is basically an extension of the scripting language Tcl, and a collection of directives of how to use the specified commands.

The power modelling concept is based on finite state machines, in this context also known as PSMs. Each state has a specific power value, either constant or computed by an equation which can be dependent to other input values, e.g. the current frequency. Transitions from one state to another are triggered by various inputs, such as logic signal switches, timer events or SystemC TLM transaction events. The latter forms the basis in this work to apply the black box constraint. As no internals can be observed in closed-source components, only the inter module communication is available. The previous works [14, 17] and Section 5.1 show that TLM transactions are sufficient for accurate power modelling. To represent this transition mechanism, the trace matrix stores just the information if a triggering event happened at a certain time, i.e. $\mathrm{S} \in \{0,1\}^{T' \times N}$. T' is the number of sampling periods t'_{samp}, which is the shortest time interval between two events.

Further, UPF can be used to partition a design into power domains. Each domain has its own PSM and input trigger set. The sum over all domains determines the power consumption of the entire MPSoC. This not only reduces the complexity of a PSM, but also eases the transformation from the linear model approach to UPF.

3.3 Calibration of UPF PSMs

For the semi-automatic calibration of UPF PSMs, the linear power model of the estimation methodology presented in [17, 18] is used as starting point. It is essential to compute the power model factors first. As shown in [17], the non-negative least-squares approach reveals the most accurate results.

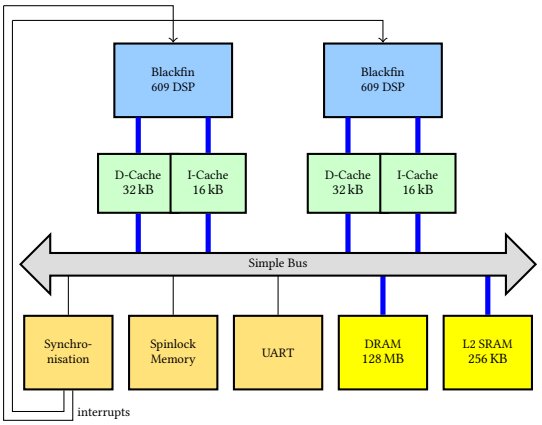


Figure 3: Virtual platform for the Blackfin 609 processor on the FinBoard

$$\mathbf{a} := \underset{\mathrm{x}}{\operatorname{argmin}} \|\mathrm{S} \cdot \mathbf{x} - \mathbf{P}_{\mathrm{ref}}\|_2 \ \text{ subject to } \ \mathbf{x} \geq \mathbf{0} \tag{3}$$

where $\mathbf{P}_{\mathrm{ref}} \in \mathbb{R}^T$ is the reference power trace recorded with an equal sampling rate at lower level than ESL or real hardware. It is worth to mention that substituting S with $\mathrm{S}' = \begin{pmatrix} \mathrm{S} & f \cdot \mathrm{S} \end{pmatrix}$ calibrates the power model factors for frequency aware power models.

To transform the linear power model into UPF compliant PSMs, each a_i needs to be translated into a separate power domain with two states. State 1 is always 0, while state 2 is the value of a_i. In the frequency aware model, state 2 is either $a_{\mathrm{fd}\,i} \cdot f$ in case of a frequency dependent factor or $a_{\mathrm{fi}\,i}$. The corresponding trigger event is the same as the tracing position of trace $\mathbf{s}_i$. State changes appear from state 1 to 2 if $s_{i,t} = 1$, and from 2 to 1 if $s_{i,t} = 0$. A simplified example of PSMs with the corresponding linear power model is shown in Figure 1. For the constant trace $\mathbf{s}_1 = \mathbf{1}$, no transitions are required. Thus, the corresponding state takes the value of the first power model factor $\mathrm{a}_{\mathrm{fi}\,1}$, and has no transitions. Trace $\mathbf{s}_2$ is frequency independent which means that there is one state with the value of $\mathrm{a}_{\mathrm{fi}\,2}$ and it is activated if $s_{2,t} = 1$. The transition to the zero state happens if $s_{2,t} = 0$. The frequency dependent PSM of trace $\mathbf{s}_3$ has to be constructed like $\mathbf{s}_2$ with the modification that if $s_{3,t} = 1$, the state has the value $f \cdot \mathrm{a}_{\mathrm{fd}\,3}$.

4 CASE STUDIES

The proposed extension for supporting standard compliant power models with the black box power estimation methodology would lack comprehensible data without using representative case studies. For better comparison with previous work, the ARM case study introduced in [17] and the Blackfin DSP case study presented in [14] are reused. In both case studies, the reference power measurement contains the power consumption of the processors and L1 cache system. Hence, the power models are built for the processors including the L1 cache system. Before summarising the case studies and employed benchmark set, it is worth to mention that tracing

has been adapted. In the previous work, counters are used for the so called *TLM trace*. This means, after every t_{samp} read and write counters are stored and then reset to zero. This work introduces the *TLM event trace*, as UPF reacts on TLM transaction events. Hence, the trace stores every write and read occurrence of a TLM transaction, which can happen every t'_{samp} and is the shortest delay possible in the VP, i.e., the time the shortest event lasts. This results in $t'_{\text{samp}} \ll t_{\text{samp}}$. To keep the size of the traces reasonable, there is an averaging step added after t_{samp}. Consequently, the percentage of how often and long a TLM event happened during t_{samp} is stored.

Further abstraction similar to the introduced *activity trace* in [17] is still possible. Instead of using TLM triggering events, a plain SystemC signal is used to indicate whether a processor is active or idle. Thus, the activity trace is set to one, if the core is running. Otherwise, the trace stores zero.

4.1 ARM Case Study

The reference system for the ARM case study is the dual-core ARM subsystem of the OMAP4460 SoC from the PandaBoard. The VP is composed around an instruction accurate ARM Cortex-A9 processor model from OVP. For this work, the instruction accurate standalone simulator contained in the gdb-utils and encapsulated in a SystemC wrapper replaces the OVP processor model. Besides the ARM, the entire memory hierarchy is assembled using an in-house virtual component model library, i.e. L1 data cache with the implementation of the coherency mechanism, L2 caches, DRAM and buses are present in the VP. All connections are modelled using TLM blocking transactions, as this is sufficient for instruction accurate processor models. Average timing errors of 9 % are reported [17]. Figure 2 illustrates the VP. The peripherals *synchronisation* and *spinlock memory* are used to implement and enable multi-threaded applications. The locations of TLM event tracing are indicated by thick blue lines.

4.2 Blackfin Case Study

As reference for the Blackfin case study, the dual core Blackfin 609 DSP from the FinBoard is taken. The instruction accurate processor model from gdb-utils is wrapped into SystemC context and used in the VP. From the same in-house virtual component model library, the memory hierarchy is built, i.e. L1 caches, L2 SRAM, DRAM and buses. TLM blocking transactions are used to model the inter-component communication. The timing error of the VP is on average 6 % [14]. In Figure 3, the locations of TLM event tracing indicated by thick blue lines can be seen. As for the ARM case study, multi-threaded applications are enabled and executed using the peripherals *synchronisation* and *spinlock memory*.

4.3 Benchmarks

The benchmarks are executed without an operating system and directly run as *bare-metal* code. A representative benchmark set has been chosen to stress and verify the methodology and reveal representative results. In-house benchmarks in addition to well-established standard benchmarks are used.

The standard benchmark selection contains: Dhrystone [26], LTE uplink receiver PHY benchmark [20] (abbreviated `lte-bench`), telecomm package of MiBench [8] (`mib/t`), StreamIt [24] (`stit`)

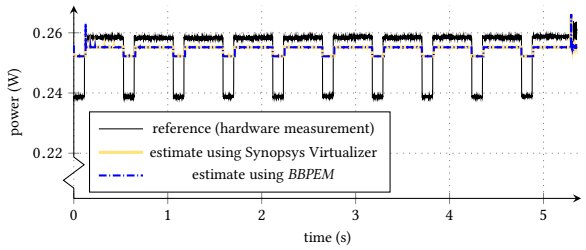


Figure 4: Reference power curve and estimated power consumption curves using TLM event traces for the `lte-bench` and Blackfin case study

and WiBench [28] (`wib`). Some of the standard benchmarks have been modified to use integer arithmetic instead of floating point emulation. In the following, their name contains the suffix `_int`. Both variants, integer and floating point arithmetic, have been used.

Additional in-house multicore benchmarks test the power modelling capabilities of the estimation methodology for the dual core processors in both case studies. Multi-thread benchmarks are named `mt`. The Dhrystone benchmark is modified to run first on both cores and then twice on one core while the other is idle. This scheme is repeated three times (`dhrystone2co`).

5 EVALUATION

The evaluation of the proposed extension to the black box ESL power estimation methodology is performed in two steps. First, it is investigated what influence the switch from counter based TLM traces to the TLM event traces has on the accuracy. For better clarification in the remainder of this paper, the name *BBPEM* is introduced. Estimates computed with TLM event traces or activity traces, and the usage of the linear power model are later referred to as received with *BBPEM*. The second evaluation step examines the export approach of UPF PSMs. Therefore, the same VPs of the ARM and Blackfin case study have been ported to Synopsys Virtualizer, which supports the IEEE 1801-2015 standard. The UPF power model generated out of the *BBPEM* flow is imported to Synopsys Virtualizer and the same evaluation procedure repeated to have comparable results.

To increase the number of testcases, the evaluation is performed using the leave-one-out cross-validation scheme: all benchmarks but one serve as training set to calibrate the power model factors and export UPF PSMs. The remaining benchmark is used to perform and evaluate the power estimation methodology.

Due to the timing error of the VP, the reference power trace and ESL traces might not have the same length. Consequently, high power consumption phases in the reference trace would be shifted compared to high activity observed in the ESL trace. To avoid this timing mismatch, the ESL traces are rescaled to the length of the reference.

5.1 Power Estimation Results

For obtaining the power estimation results, the relative root mean squared (RMS) of the error over time of estimated and the reference

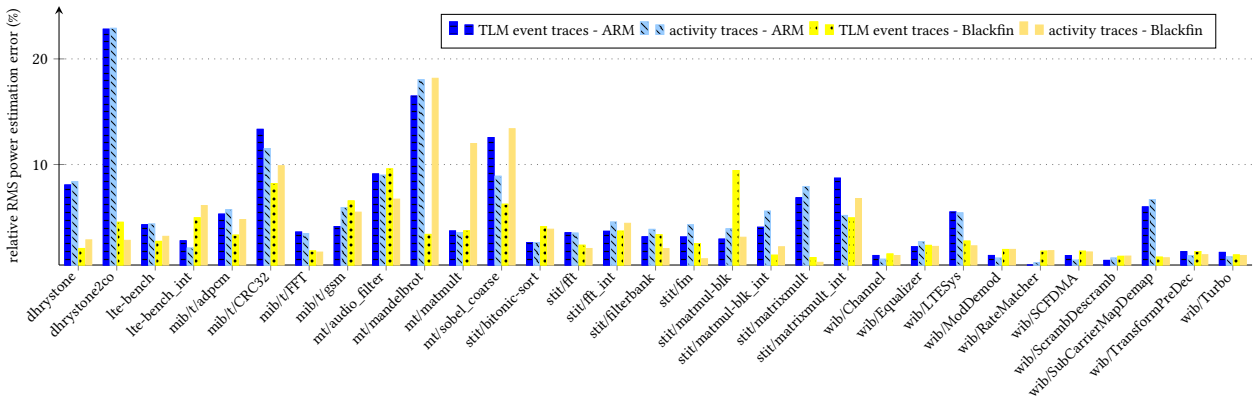


Figure 5: Relative RMS power estimation error for *fixed frequency* power models and *BBPEM*

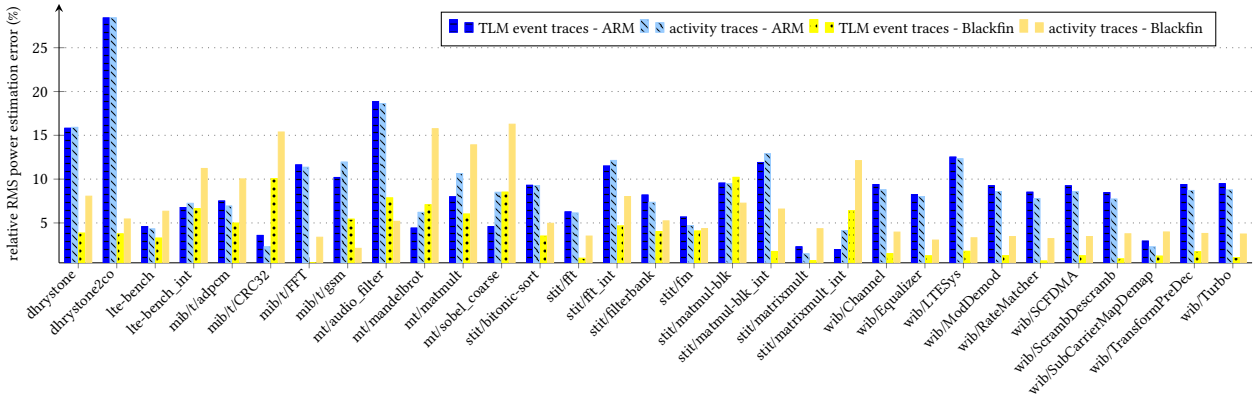


Figure 6: Relative RMS power estimation error for *frequency aware* power models and *BBPEM*

power has been used:

$$e_{\text{RMS}} = \frac{\sqrt{\frac{1}{T} \cdot \sum_{t=1}^{T} \left(P_{\text{est},t} - P_{\text{ref},t}\right)^2}}{\overline{P_{\text{ref}}}} \tag{4}$$

The resulting RMS error of each estimation is shown in Figure 5 for the fixed frequency model using Equation 1. There are four bars per benchmark. The two left blue coloured ones belong to the ARM case study and the right two yellow bars to the Blackfin case study. Over all, it can be seen that out of the 31 benchmarks, just four have a higher error than 10 %, with the maximum error of 22.8 %. Further, Blackfin power estimates are on average more accurate. However, these results show that the proposed methodology extension is well applicable in the UPF context. Differentiating between the TLM event and activity trace methods, the average errors are 4.4 % and 4.8 %, respectively. This proves that TLM event traces reveals slightly more accurate power estimation with at cost of more tracing overhead. And activity trace based power models provide an excellent trade-off between tracing effort and estimation accuracy.

A comparison of the reference power curve and estimated power consumption curve using TLM event traces for the `lte-bench` and the Blackfin case study can be seen in Figure 4. At first, the coarse structure is reproduced by the estimate closely, though the amplitude does not match exactly. Second, the fine structure of the reference measurement can be found partly in the estimate.

For the assessment of the frequency aware linear power model, the relative power estimation RMS error is plotted in Figure 6. First of all, an increase of the error can be observed compared to the fixed frequency approach. The average error for TLM event and activity tracing is increased to 6.4 % and 7.8 %, respectively. The maximum error is 28 %. An explanation for the larger error are non-linearities present when allowing the frequency to be changeable. However, the simple linear model fails to compensate this.

5.2 Framework Comparison

The generated UPF models need to be verified by means of a second tool that officially implements the standard. This is necessary to ensure that the export approach is reliable enough to exchange the power models. Therefore, they are generated out of the *BBPEM* flow and imported to Synopsys Virtualizer. Repeating the same evaluation procedure as in the previous section delivers the results visualised in Figure 7, together with the relative RMS power estimation error possible using *BBPEM*. Due to the huge amount of

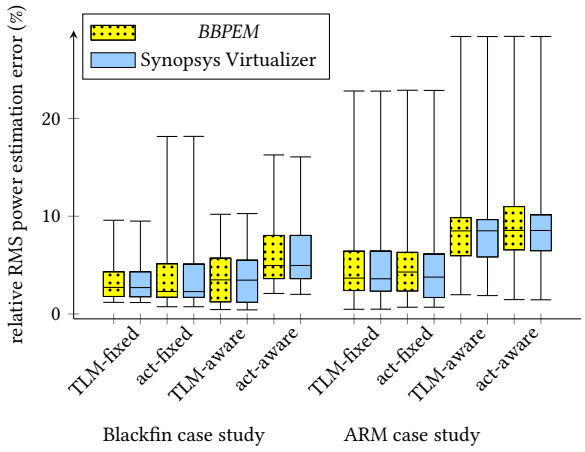


Figure 7: Power estimation error comparison for *BBPEM* and Synopsys Virtualizer

testcases, just an overview in the form of a box-and-whiskers plot is given. The left-hand side of the figure shows the results for the Blackfin case study and the ARM case study can be found on the right-hand side. For each tracing variant (TLM event and activity traces) and power model (fixed frequency and frequency aware), the left yellow box shows the results of *BBPEM*. The right blue box represents the outcome of Synopsys Virtualizer. Comparing the minimum and maximum (the lower and higher whisker, respectively) directly reveals almost no differences. The 25 % and 75 % percentile (lower and upper bound of the box, respectively) and median (bar inside the box) show again only marginal differences. With these findings, the linear power modelling approach can be translated into UPF PSMs without loosing accuracy. Additionally, it is applicable to exchange the power models between two different frameworks, while achieving the same results.

6 CONCLUSIONS

A method for enabling semi-automatic calibration of IEEE 1801-2015 standard compliant system level PSMs is presented in this work. The extension of the black box ESL power estimation methodology of [17] is proven effective with two representative case studies. First, the traceable information are converted from counter based TLM traces to standard compliant TLM event traces. Second, a translation from the linear power model originally used in this methodology into UPF PSMs is proposed. Two case studies have been conducted, one with an ARM Cortex-A9 and the other with a Blackfin 609 DS. Average estimation errors of about 4.6 % and 7.1 % can be observed using fixed frequency and frequency aware power models, respectively. Further, it is shown that the generated PSMs out of the *BBPEM* flow can be imported in Synopsys Virtualizer without any significant difference in the power estimation accuracy.

REFERENCES

[1] Docea Aceplorer. [Online] `https://www.intel.com/content/www/us/en/system-modeling-and-simulation/docea/overview.html` (accessed 10/2017).
[2] IEEE standard for design and verification of low-power, energy-aware electronic systems. pages 1–515, March 2016.
[3] SystemC. [Online] `http://www.accellera.org/downloads/ standards/systemc` (accessed 09/2015).
[4] T. Bouhadiba, M. Moy, and F. Maraninchi. System-level modeling of energy in TLM for early validation of power and thermal management. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, 2013.
[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, June 2000.
[6] L. Eeckhout and K. D. Bosschere. Early design phase power/performance modeling through statistical simulation. In *Proc. 2001 IEEE Intl. Symp. on Performance Analysis of Systems and Software*. IEEE, 2001.
[7] D. Greaves and M. Yasin. TLM POWER3: Power estimation methodology for SystemC TLM 2.0. In *Proceedings of the 2012 Forum on specification & Design Languages*, Semptember 2012.
[8] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, WWC-4*. IEEE CS, 2001.
[9] W.-T. Hsieh, J.-C. Yeh, S.-C. Lin, H.-C. Liu, and Y.-S. Chen. System power analysis with DVFS on ESL virtual platform. In *SOC Conference (SOCC), 2011 IEEE International*, 2011.
[10] M. Jung, K. Kraft, and N. Wehn. A new state model for DRAMs using petri nets. In *IEEE International Conference on Embedded Computer Systems Architectures Modeling and Simulation (SAMOS)*, July 2017.
[11] M. Jung, C. Weis, P. Bertram, G. Braun, and N. Wehn. Power modelling of 3D-stacked memories with TLM 2.0 based virtual platforms. In *Synopsys User Group Conference (SNUG)*, May 2013.
[12] S. Kumar Rethinagiri, O. Palomar, J. Arias Moreno, O. Unsal, and A. Cristal. VP-PET: Virtual platform power and estimation tool for heterogeneous MPSoC based FPGA platforms. In *Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2014.
[13] J. Laurent, N. Julien, E. Senn, and E. Martin. Functional level power analysis: an efficient approach for modeling the power consumption of complex processors. In *Design, Automation and Test in Europe Conference and Exhibition*, 2004.
[14] G. Onnebrink, S. Schürmans, F. Walbroel, R. Leupers, G. Ascheid, X. Chen, and Y. Harn. Black box power estimation for digital signal processors using virtual platforms. In *RAPIDO '16 workshop*, 2016.
[15] S. Pasricha, Y.-H. Park, F. Kurdahi, and N. Dutt. System-level power-performance trade-offs in bus matrix communication architecture synthesis. In *Hardware/-Software Codesign and System Synthesis.*, 2006.
[16] S. K. Rethinagiri, O. Palomar, R. Ben Atitallah, S. Niar, O. Unsal, and A. C. Kestelman. System-level power estimation tool for embedded processor based platforms. In *Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, 2014.
[17] S. Schürmans, G. Onnebrink, R. Leupers, G. Ascheid, and X. Chen. Frequency-aware ESL power estimation for ARM Cortex-A9 using a black box processor model. ACM Transactions on Embedded Computing Systems (TECS), 2016.
[18] S. Schürmans, D. Zhang, D. Auras, R. Leupers, G. Ascheid, X. Chen, and L. Wang. Creation of ESL power models for communication architectures using automatic calibration. In *Design Automation Conference (DAC)*, 2013.
[19] G. Shalina, T. Bruckschloegl, P. Figuli, C. Tradowsky, G. Almeida, and J. Becker. Bringing Accuracy to Open Virtual Platforms (OVP): A Safari from High-Level Tools to Low-Level Microarchitectures. volume ICIIIOES, pages 22–27, Dec. 2013.
[20] M. Själander, S. McKee, P. Brauer, D. Engdal, and A. Vajda. An LTE uplink receiver PHY benchmark and subframe-based power management. In *Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2012.
[21] E. Sotiriou-Xanthopoulos, G. Percy Delicia, P. Figuli, K. Siozios, G. Economakos, and J. Becker. A power estimation technique for cycle-accurate higher-abstraction systemc-based cpu models. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015*, 2015.
[22] M. Streubühr, R. Rosales, R. Hasholzner, C. Haubelt, and J. Teich. ESL power and performance estimation for heterogeneous MPSoCs using SystemC. In *Specification and Design Languages (FDL)*, 2011.
[23] Synopsys Virtualizer. [Online] `http://www.synopsys.com/prototyping/virtualprototyping/pages/virtualizer.aspx` (accessed 10/2017).
[24] W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A language for streaming applications. In *Intl. Conference on Compiler Construction*, Grenoble, France, 2002.
[25] Y. Veller and S. Matalon. Why you should optimize power at the ESL – Whitepaper, Mentor Graphics. [Online] `http://go.mentor.com/cvtq` (accessed 10/2017), Aug 2010.
[26] R. P. Weicker. Dhrystone: A synthetic systems programming benchmark. *Comm. ACM*, 1984.
[27] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of Simplepower: A cycle-accurate energy estimation tool. In *Proceedings of the 37th Annual Design Automation Conference*, 2000.
[28] Q. Zheng, Y. Chen, R. Dreslinski, C. Chakrabarti, A. Anastasopoulos, S. Mahlke, and T. Mudge. WiBench: An open source kernel suite for benchmarking wireless systems. In *Workload Characterization (IISWC)*, 2013.

Beyond Time-Triggered Co-simulation of Cyber-Physical Systems for Performance and Accuracy Improvements

Beyond Time-Triggered Co-simulation of Cyber-Physical Systems for Performance and Accuracy Improvements.

Giovanni Liboni
Universite Cote d'Azur, Inria
Sophia Antipolis, France
giovanni.liboni@inria.fr

Julien Deantoni
Universite Cote d'Azur,
CNRS, I3S, Inria
Sophia Antipolis, France
deantoni@polytech.unice.fr

Antonio Portaluri
EDALab Srl
Verona, Italy
antonio.portulari@edalab.it

Davide Quaglia
Department of Computer Science,
University of Verona, Italy
davide.quaglia@univr.it

Robert de Simone
Universite Cote d'Azur, Inria
Sophia Antipolis, France
robert.de_simone@inria.fr

ABSTRACT

Cyber-Physical Systems consist of cyber components controlling physical entities. Their development involves different engineering disciplines, that use different models, written in languages with different semantics. A coupled simulation of these models is of prime importance to rapidly understand the emerging system behavior. The coupling of the simulations is realized by a coordinator that conveys data and ensures time consistency between the different models/simulators. Existing coordinators are usually time triggered. In this paper we show that time-triggered coordinators may introduce poor performance and accuracy (both temporal and functional) when used to co-simulate cyber-physical models. Therefore, we propose a new coordinator mixing time- and event-triggered mechanisms. We validated the approach in the context of the FMI standard for co-simulation. To make possible the writing of the new coordinators, we implemented backward compatible extensions to the FMI API. Also, we implemented a new FMI exporter in an industrial tool.

KEYWORDS

FMI, Functional Mock-up Interface, event-driven simulation

ACM Reference Format:
Giovanni Liboni, Julien Deantoni, Antonio Portaluri, Davide Quaglia, and Robert de Simone. 2018. Beyond Time-Triggered Co-simulation of Cyber-Physical Systems for Performance and Accuracy Improvements.. In *RAPIDO: Rapid Simulation and Performance Evaluation: Methods and Tools, January 22–24, 2018, Manchester, United Kingdom.* ACM, New York, NY, USA, Article x, 8 pages. https://doi.org/10.1145/3180665.3180668

RAPIDO, January 22–24, 2018, Manchester, United Kingdom

ACM ISBN 978-1-4503-6417-1/18/01…$15.00
https://doi.org/10.1145/3180665.3180668

1 INTRODUCTION

In Cyber-Physical Systems (CPS) digital systems (eventually distributed) are in charge of controlling physical entities in the environment, e.g., mechanical, chemical or thermal mechanisms. Modeling and simulation are traditional techniques to support design but in CPS's there is a mix of different engineering disciplines [18] each of them using a domain-specific modeling language with its syntax and semantics. The simulation of such systems should address a mix of heterogeneous models whose coupling make the behavior of the system under modeling emerging [8]. To ensure a fast time to market, it is important to understand such emerging behavior early in the development process.

In this context, co-simulation is a key enabler. It consists in the coordinated simulation of different parts of the CPS by using specific solvers/interpreters for the different modeling approaches. The "coordinator" among the simulators must ensure the timely exchange of data between the different solvers/interpreters according to a so-called "master algorithm". The development of such coordinator usually relies on the graph representing the sharing of data between the different models [2, 4, 5, 24]. It also depends on the nature of the interaction between the different models [10].

For instance, when the data shared between two models have a causal relationship (*e.g.*, a producer-consumer relationship) then the coordinator must ensure that it is respected during the co-simulation. In literature, there are coordination languages to define a partial order between the execution of the models [22].

This coordinator is crucial for the correctness and efficiency of the co-simulation so that many master algorithms have been proposed [1, 2, 4, 5, 21, 23, 24, 27, 28]. It is worth noticing that all the proposed algorithms are *time triggered*, i.e., model execution is forced at predefined

time-steps. This is a surprising fact since from the 90's work about coordination languages and architecture description languages (ADLs) proposed more sophisticated techniques for the correct and efficient coordination among software components [13, 19, 22].

In this paper, we highlight the loss of efficiency and accuracy introduced by the use of pure time-triggered coordinators by presenting minimal illustrating examples. Then, we leverage work done in the community to introduce new master algorithms, especially relevant for CPS since they mix event and time trigger mechanisms. Finally, we present experimental results for co-simulation based on the FMI [20] industrial standard where VHDL digital models are coordinated with Modelica physical models. To enable the writing of new coordination algorithm, we extended the FMI standard in a backward compatible way.

2 BACKGROUND ON TIME-TRIGGERED COORDINATORS

The most common coordinator runs each model for a step (*i.e.*, a fixed period of time), collects the outputs from all subsystems and conveys outputs to the inputs of the model(s) of interest. Finally, it continues the (co-)simulation for the required simulation time [1].

In a model, when a discontinuity appears during the simulation of a step, it may *reject the step* or not depending on its implementation. If the step is rejected, the model returns the discontinuity time to the coordinator and all the models that already simulated the current step need to be *rolled back* to their previously saved state. Then they are asked to simulate until the time of the discontinuity, values are exchanged and the simulation continues in a time-driven fashion. If the step is not rejected, then the exact time of the discontinuity is unknown; it occurred during the step. Note that implementation of the rollback procedures is costly, not always implemented, and according to [9], it may be difficult to achieve in practice, especially for cyber models.

To understand how to write a coordinator, it is important to understand that cyber and physical models are fundamentally different in the way they are executed (*i.e.*, they follow a different Model of Computation). These differences have already been identified and studied in the context of co-simulation [4, 5, 26]. However, the goal of these studies was not to propose an appropriate coordinator but rather to proposed functional and temporal adaptation considering these differences.

Physical models are usually defined by equations (*e.g.*, ODE or DAE) which can be solved at any points in time and a numerical approach is used to choose the most appropriated discretization. Usually, in such physical models, smaller the discretization step is, better the accuracy is.

[1] many variants of this simple abstraction are available in the literature [1, 2, 4, 5, 21, 23, 24, 27, 28] but the main idea is the same.

Cyber models are based on the notion of, possibly parallel, sequences of instructions and data are read or written at specific points in time which depend on the program structure and are not necessarily periodic. So it is not possible to use a numerical method to compute an appropriate constant step size. In the context of time-triggered simulation, each data transfer can then be seen as a discontinuity and may involve step rejection with state saving/restoring thus inducing a possible large overhead at runtime [9]. When step rejection is not used it implies accuracy problems (see Section 3).

In summary, the designer can either use step rejection (when implemented) or reduce the co-simulation step; in both cases there is a loss in simulation speed [6]. If the co-simulation step is increased, simulation accuracy is reduced. Therefore, a time-triggered coordinator in the presence of cyber model(s) forces to have a trade-off between performance and accuracy.

3 MIX EVENT- AND TIME-TRIGGERED COORDINATOR FOR CPS

3.1 Overview

We want to enable the use of cyber models for the co-simulation of cyber-physical systems. Based on previous experiments on synchronous languages [3] and the coordination of heterogeneous cyber models [7, 16, 17] we believe that coordinators could be more accurate and efficient if we allow event-driven communication between the coordinator and the models under execution. Our goal is twice. On the one hand, we want to improve the efficiency of the simulation by 1) avoiding roll back as much as possible and 2) reducing the communication between a model under simulation and the coordinator. On the other hand, we want to improve the accuracy (both temporal and functional) by letting a model simulate until an event of interest occurs.

This is a well-known mechanism in distributed systems where unnecessary communications are avoided and where logical clocks are used to synchronize the systems [11, 15]. This means that the coordinator must be tuned, according to both the data sharing topology and their properties to avoid as much as possible the unnecessary synchronizations between the model simulations, letting them simulate until a communication is required.

In the next subsections, we describe three examples that highlight a particular problem within the current time-triggered coordinators. We also propose a coordinator to solve the problem[2].

[2] All the experiments presented in this paper, the results and the implementations are provided here https://project.inria.fr/fidel/rapido2018/

3.2 Coordination of a cyber model with discrete output

The first example shows limitations of the time-triggered coordination for a cyber model with a discrete output.

Let us consider the cyber model of a wheel encoder driver ($C1$), producing a discrete signal $v1$. A wheel encoder is a sensor that allows tracking the number of wheel rotations. More precisely, the output of the driver switches from 0 to 1 or conversely each time a 1/32 of revolution of a wheel is done. Signal $v1$ is consumed by another model ($P1$) (see Figure 1), which computes the actual speed of the wheel according to the time spent between two successive switches of $v1$. Consequently, $v1$ is assigned only at specific points in time, depending on the speed of the wheel. This assignment creates a discontinuity, which is neither a rare event or symptomatic of a specific phenomenon like in models of physical systems.

In order for the co-simulation to be correct, all data assignments should be seen by the coordinator and transferred to $P1$ at the right time. More generally, such discontinuity implies different problems: a temporal inaccuracy, *i.e.*, the changes in the output values are seen by the coordinator at wrong points in time; a functional inaccuracy, *i.e.*, the coordinator is missing some of the discontinuities; or a performance problem, *i.e.*, the roll back mechanism is used intensively.

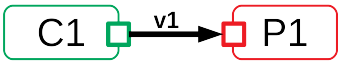


Figure 1: Model $C1$ produces data $v1$ consumed by $P1$.

In order to illustrate this phenomenon, we have created different time-triggered co-simulation runs where the speed of the wheel is constant (*i.e.*, $v1$ changes periodically). We have used four different setups for the co-simulation period and we have studied the corresponding impact on the speed computation according to the information retrieved by the coordinator. The results are shown in Figure 2, where we can see that smaller the time trigger period is, smaller the error is. However, this is never perfect due to the sampling of the coordinator. Additionally, this oversampling leads to useless communication points which decrease performance (see [6]), a fortiori when the wheel turns slowly (*i.e.*, when the period of $v1$ is bigger). If the model simulator supports step rejection, then the accuracy problem does not hold. However, it introduces roll-backs for each 1/32 of wheel revolution and consequently a performance problem (see [9]). Finally, when the time between two switches in the output of the wheel encoder driver is not constant, the period used for the co-simulation is either pessimistic or introduce inaccuracy.

To overcome these problems, we propose to let the simulator decide when the simulation must be stopped according

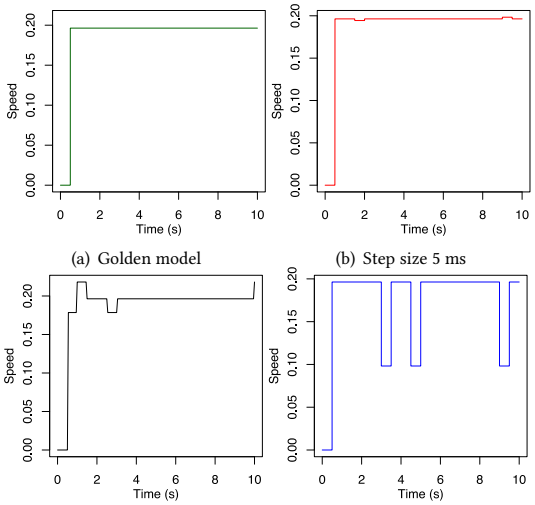


Figure 2: Comparison of the wheel speed between the golden model and time driven coordinator with different communication step sizes.

to the configuration done by the coordinator. The goal is to avoid step rejection, to obtain good accuracy and to reduce as much as possible the communications between the model simulator and the coordinator.

Algorithm 1 Coordination algorithm involving a discrete output

```
1:  P1 := newSpeedComputationModel;
2:  C1 := newWheelEncoderModel;
3:  while t ≤ t_end do
4:      C1.simulateUntilDiscontinuity(
5:       v1, &t_nextEvent);
6:      P1.doStep(t, t_nextEvent − t);
7:      tmp = C1.getV1();
8:      P1.setV1(tmp);
9:      t := t_nextEvent;
10: end while
```

To drive the co-simulation, we used the coordination specified in Algorithm 1, where the simulation of $C1$ and $P1$ is coordinated by using both the traditional time-triggered service (*doStep*) and a new event-based service (*simulateUntilDiscontinuity*). Its interface is defined as follows.

```
simulateUntilDiscontinuity(
    in Set<Variable> monitoredVars,
    out time nextEventTime)
```

When using this function on a specific model, the model simulator monitors the assignments of each variable in the `monitoredVars` set. Every time a discontinuity happens on

one of these variables, the function sets `nextEventTime` to its current internal time and returns immediately. The function takes two parameters:

- The *monitoredVars* input parameter is the list of variables the model simulation has to monitor, looking for a discontinuity;
- The *nextEventTime* output parameter returns the internal simulation time when a discontinuity occurred.

The main idea of the coordinator algorithm is that the coordinator asks $C1$ to simulate until a discontinuity is detected on $v1$. When the function returns, the coordinator knows 1) the exact time of the assignment and 2) that this is the first assignment of interest that happened between time t and $t_{nextEvent}$. Then the coordinator calls the *doStep* function on $P1$ (line 6) to simulate it until the time at which the discontinuity appeared in $C1$. Then, it retrieves the value that caused the discontinuity from $C1$ and set it to $P1$ (line 7 and 8). By using this coordination algorithm, the number of communication points is equal to the number of discontinuities in the cyber model (*i.e.*, the smallest one to retrieve all the values of the variable) and the timestamps of the discontinuities are precisely known. This simple coordination provides no rollback, no overhead in the co-simulation execution time and perfect temporal accuracy.

3.3 Coordination of a cyber model with input(s)

As a second example, consider a cyber model $C1$ that senses the environment (*e.g.*, a room, a CPU) and computes the temperature. Usually, in the actual implementation of such system, the environment is sensed periodically. We consider here that the environment that provides the temperature evolution is modeled by a physical model $P1$ whose output is read periodically by $C1$ (see Figure 3). In usual time-triggered co-simulation, the co-simulation period is chosen so that the data obtained by the physical model is *fresh* enough when propagated to the cyber model.

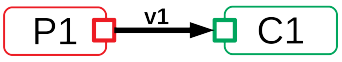


Figure 3: Model $P1$ produces data $v1$ consumed by $C1$.

There are three drawbacks here. First, the cyber model is called several times to update its input even if this input is not required to be read internally thus wasting simulation time. Second, the physical model is called several times to compute fresh values that are actually not used by the cyber model. Third, there is no synchronization between the actual reading of the input by the cyber model and its update by the coordinator. This can lead to a temporal inaccuracy since the actual reading can occur at the end of a simulation step, *i.e.*, without a *fresh* input. In this case, either the designer considers that the freshness of the data is not important (but that can lead to wrong simulation results !) or the designer decreases the co-simulation period and consequently decreases the performance.

As shown in the previous section, increasing the number of communication points between models and coordinator for better accuracy decreases the overall performance and therefore we aim at reducing the number of communication points without reducing accuracy.

To solve this problem, we propose to enable the simulation of a model until the precise time when it is ready to internally read on one of its inputs. It provides the coordinator with the time at which the read operation will be actually done. On the one hand it avoids unnecessary calls to the cyber model simulator; on the other hand, the coordinator can ask the physical model to compute the input data at the exact time it will be read by the cyber model (by calling the traditional `doStep` method with the appropriate communication step size). At the next call of the cyber model, it will read the input data that has been updated specifically for it.

Algorithm 2 describes the coordinator that implements such proposition. In this case, $C1$ is simulated first (line 4 and 5). When it returns, $P1$ is simulated until the reading time of $C1$ (line 6). Then the value retrieved from P1 is sent to C1 (line 7 and 8).

Algorithm 2 Co-simulation Master Algorithm read operation

```
1:  P1 := newEnvFMU;
2:  C1 := newSensorDriverFMU;
3:  while t ≤ t_end do
4:      C1.simulateUntilRead(
5:       v1, &t_nextEvent);
6:      P1.doStep(t, t_nextEvent − t);
7:      tmp = P1.getV1();
8:      C1.setV1(tmp);
9:      t := t_nextEvent;
10: end while
```

To create this coordination we used a new event-based interface named *simulateUntilRead*, defined as follows:

```
simulateUntilRead(
    in Set<Variable> inVars,
    out Time& nextEventTime)
```

It enables the simulation of a model until it is ready to do a read operation on one of the input variables in the `inVars` set. The function takes two parameters:

- The `inVars` input parameter is a list of *sensitive* Variable for which the function should return before their communication;
- The `nextEventTime` output parameter provides the internal simulator time just before the read occurs.

3.4 Conditional simulation

Sometimes, input values of a model internally participate in a conditional statement. Depending on the condition, different behaviors are chosen (*e.g.*, by using a traditional `if` statement). For instance, let us consider a simple counter $C1$ that increments a value $v1$ consumed by a model $P1$. Internally to $P1$, when $v1$ reaches a specific value, then $P1$ changes its behavior. Usually, this is implemented in co-simulation by periodically providing the input value to the model, which checks if the value reaches the condition or not. To avoid previously presented drawbacks, a designer could use the coordinator presented in section 3.2 where the input value is provided with a good temporal precision. We go further here by defining a coordinator that asks a model to simulate until a specific condition is reached on one of its output variables. This avoids unnecessary communication points between the model simulators and the coordinator and consequently provides better performance.

The coordinator used in this case (Algorithm 3) is similar to Algorithm 1. However, during the setup phase, it retrieves the predicates from $P1$ to construct the condition variables sent as a parameter of $C1$ simulation.

Algorithm 3 Coordination Algorithm with Predicate

```
1: C1 := newEnvFMU;
2: P1 := newSensorFMU;
3: conds := P1.getPredicates();
4: condVars := setupVars(conds);
5: while t ≤ t_end do
6:     C1.simulateUntilCondition(
7:      condVars, &t_nextEvent);
8:     ... (see Algorithm 1)
9: end while
```

This coordinator uses a new event-based interface to handle conditional checks:

```
simulateUntilCondition(
    Set<CondVariable> outVars,
    Fmu2Time& nextEventTime)
```

`CondVariable` extends the `Variable` structure with a Boolean predicate. Using this interface, the model providing data is simulated and each time an assignment is done on one of the variables in `outVars`, then the predicate is evaluated. When a predicate is evaluated to `True`, then the function returns. This interface can have a positive effect on performance provided that more information about the model internal behavior are available with respect to traditional co-simulation.

4 TOOL INTEGRATION

In this section, we present the technical background used for the experiments. We are going to present FMI, the industrial co-simulation standard, and HIFSuite, a tool suite to perform model manipulation.

4.1 FMI - Functional Mock-up Interface

FMI is a tool-independent standard framework for co-simulation of dynamic models. The FMI standard is managed and developed as a Modelica Association Project. FMI provides a standardized interface allowing different executable models (named FMU: Functional Mock-up Unit) to be controlled by an external software entity. The data exchange between models is restricted to communication points. Between two communication points (i.e., during a simulation step), the models are solved independently by each FMU simulator. The coordination is implemented by the so-called *Master Algorithm* (MA). FMI standard regulates the set of interfaces provided by each simulator (i.e., FMU) which are called by the master algorithm. The master algorithm is not part of the FMI standard. It can *set* or *get* the current value of an exposed variable (according to its direction) by using the standardized FMI API. This API is also used to simulate the model for a specific interval of time specified in the *doStep* method. In the co-simulation mode, each FMU solver decides how many computational steps should be done in that time interval to reach the desired precision.

In order to perform experiments, we implemented the proposed API as a backward compatible extension of FMI. The extended version of FMI is available on the already mentioned website.

4.2 HIFSuite

In order to drive our experiments based on both the existing and the extended FMI API, we need cyber models for which it is possible to adapt the FMU wrapper. We adopted EDALab's HIFSuite[3] to make the process automatic.

HIFSuite provides tools to automatically perform sophisticated manipulations on models written in state-of-the-art Hardware Description Languages (HDL), like Verilog and VHDL. An HDL file is translated into a Heterogeneous Intermediate Format (HIF) description that can be manipulated by other HIFSuite tools that generate functionally equivalent C/C++ models. For our purpose, we implemented a manipulation tool to add an FMI wrapper around the model before being exported into C/C++. The resulting source code can then be used to generate an FMU as shown in Figure 4.

More in detail, a front-end tool parses the input HDL file. It analyzes all the dependencies between processes in order to handle descriptions involving both synchronous and asynchronous processes. A dependency graph is generated to reproduce the cycle-accurate behavior of the model. Also, when an HDL unit features an input clock signal, a clock generator process, which simulates the clock signal, is instantiated to make the model self executable. Then the

[3] https://www.hifsuite.com/tools

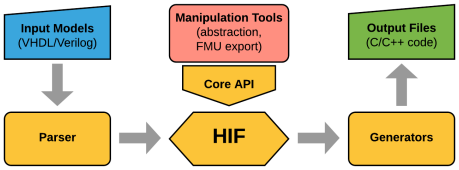


Figure 4: FMU generation flow with HIFSuite.

model is abstracted to C/C++ and all processes and data types are converted. A special structure is created to contain a field for each input and output port of the original model. At this point, the FMU exporter tool generates the wrapper that implements the FMI Standard API. For instance, the implementation of the *doStep* method ensures that the internal clock cycles reflect the required simulation time.

The last step generates the C++ source code which must be compiled to produce a shared library and compressed to become an FMU. The process has been validated by importing and testing the resulting FMU into Simulink.

To support the new interfaces, HIFSuite has to generate the previously presented methods and to modify the model behavior, *e.g.*, to stop the execution before reading a port.

5 EXPERIMENTAL RESULTS

To validate our proposal, we created three different small illustrative examples. Each example uses FMI to co-simulate two FMUs, one is a cyber FMU written in VHDL and the other one is a physical FMU written in Modelica and exported using JModelica[4]. We extended FMI introducing the new event-based interfaces presented in section 3.

5.1 Case study #1

The first case study reproduces the system described in Section 3.2, where a cyber FMU $C1$ implements a wheel encoder driver and where a physical FMU $P1$ computes the wheel speed based on the time between the value switches of the wheel encoder. The first immediate result was that we obtain a realistic value of the wheel speed from the coordinator and so $P1$ computed perfectly the wheel speed, both in the case of a constant speed and in the case of a time-varying speed (see the website for numerical results). We also obtained a number of communication points between the coordinator and the FMU which was equal to the number of switches of $v1$ so that the implementation behaved as expected.

To compare both approaches from a performance point of view, we set up the time-triggered co-simulation in order to obtain less than 1% of error on the computed wheel speed. The error with time-triggered coordinator and a wheel with a diameter of 1 meter is characterized by equation 1:

$$error = 1 - \frac{\frac{\pi}{32 \cdot \Delta t}}{\frac{\pi}{32 \cdot (\Delta t - 2 \cdot T_{TTCS})}} \quad (1)$$

where Δt is the time between two switches of $v1$ and T_{TTCS} is the time triggered co-simulation period. In a few words, the switch can occur just after a read occurs and the next switch just after the read occurs, providing a maximum temporal error of 2 times T_{TTCS}. According to this, for a switch period of 100ms (corresponding to the maximum speed of the wheel) it is required to set T_{TTCS} to 500ns. With such settings and for a simulated time of 1000 seconds we obtained the results shown in Table 1. By reducing the number of co-simulation points, the co-simulation speed is doubled.

Method	Time Triggered	Mixed
Communication Points	2000000	10000
C1 execution time (ms)	5805	4186
P1 execution time (ms)	2455	89

Table 1: Performance comparison between a time-triggered approach with less than 1% error and the proposed approach.

Method	Time Triggered			Mixed
T_{TTCS} (ms)	5	10	30	
Communication Points	2000	1000	334	200
$C1$ execution time (ns)	2123	2309	1491	1667
$P1$ execution time (ns)	4058	2446	850	529

Table 2: Comparison between the time triggered approach with difference co-simulation period and the proposed approach.

5.2 Case study #2

We implemented the case study described in Section 3.3, *i.e.*, a system composed by a physical FMU that implements the temperature of an environment and a cyber FMU that implements a sensor driver. It reads periodically the value of the environment and then uses it for further computation.

We simulated this system using different periods for the time triggered co-simulation and using the proposed API. We obtained the best accuracy with the proposed approach and the performance results for 10 seconds of simulated time are summarized in Table 2.

As in the previous experiments, with a time-driven approach, depending if we want to privilege temporal accuracy

[4]http://www.jmodelica.org

Method	Time Triggered	Mixed
Communication Points	20000	40
C1 execution time (*ns*)	4257	1653
P1 execution time (*ns*)	49682	479

Table 3: Comparison between the time triggered and the proposed approach an input is used in a conditional statement.

or performance, we have to choose a different value for the co-simulation period. If we want good temporal precision with the time triggered approach, we have to choose a small period and there is a performance drawback due to the high number of communication points. If we want to improve performance, we have to increase the co-simulation period and we obtain less accurate results due to temporal inaccuracy. In table 2, we see that the proposed approach has the minimum communication points and like previously, the new approach has the minimum cost and a perfect accuracy.

5.3 case study #3

In the third case, we implement the coordinator described in Section 3.4. In this experiment a cyber FMU implements a modulo counter, which is incremented periodically and reset to 0 at some points. Then, a physical FMU reads this value and change the slope of its output accordingly if the value is greater than a specific value or not. As described in 3, the coordinator starts by simulating $C1$, which runs until the condition expected by $P1$ is reached. Then the coordinator retrieves the value from $C1$ and simulates $P1$ until the time at which the value changes. Then the value is set to $P1$ and the process continues. Table 3 shows results for a simulated time of 10 seconds, using the proposed approach and the time-triggered one. The results are as expected. By reducing the number of communication points, we increase performance, here by a factor of 25. Note that with our mechanism, the point in time when the predicate becomes true is computed by the value provider and is consequently more accurate than with a time driven-triggered approach.

6 RELATED WORK

We are not the only ones to spot some limitations in FMI. For instance, [5] and [4] proposed to add static information in FMI (*e.g.*, input/output dependencies) or to add the event type, which has the specificity to be defined only at specific points in time. These extensions are of great interest and complementary to ours. However, they did not provide any implementations or benchmark. Some other related works like [25] proposed extensions to better integrate cyber models in a co-simulation. They proposed four new primitives among which one is close to ours: *fmi21DoStep(stepSize, nextEventTime)* where a FMU can be simulated for a specific amount of time and can stop if an internal event occurs, without a need to roll back. It is the same global idea than *fmi2SimulateUntilDiscontinuity*; however, we believe it is important to specify the variables we want to monitor since some of them can be unused or their discontinuity can be irrelevant for the system accuracy. All the other primitives introduced in [25] are either optimizations like the possibility to cancel a running step (if another rejected a step) or used when the FMU can predict their future (like already proposed in [4]). Compared to all these approaches we went a step forward by identifying situations where our new extensions make simulation faster and more accurate. Also, we implemented them in a backward compatible way. We found on a FMI forum a proposal to add discrete states and time events in FMI(https://trac.fmi-standard.org/ticket/353). Their goal was to add information about the FMU to enable the writing of a better master algorithm. We believe that such *greyification* of the FMUs are mandatory to help the designers (and eventually compilers) to better exploit FMU specificities. Unfortunately, their propositions consider mainly time-driven information. Still, the general idea is very interesting and may be completed to provide information allowing to choose between the time-driven API, the event-driver API or a combination of both. Finally, in order to improve performances, some papers proposed to distribute the FMUs on different hosts [12, 14]. The idea is interesting but according to previous tries on distributing time-driven simulations, we believe that introducing an event-driven API is a key enabler for a correct and efficient distributed simulation (as highlighted for years by [11, 15]).

7 CONCLUSION

We highlighted some performance and accuracy problems of time-triggered coordinators when used to co-simulate Cyber-Physical Systems. We proposed mixed event- and time-triggered coordinator to overcome these problems. We showed the benefits of the proposed approach on some small examples by developing a backward compatible extension of the FMI standard. Based on the model manipulation flow provided by the HIFSuite tool, we also implemented an FMI exporter for VHDL cyber models. We have multiple plans for future work. Technical future work will consider the full automation of the VHDL to FMU export and the improvement of the FMU wrapper implementation. Scientific future work will investigate a modeling environment that provides enough information about FMUs and their exposed variable to enable the automatic synthesis of the most efficient coordinator. For this purpose, we are currently exploring the use of a dedicated language based on a mix between logical and physical time.

REFERENCES

[1] Muhammad Usman Awais, Peter Palensky, Atiyah Elsheikh, Edmund Widl, and Stifter Matthias. 2013. The high level architecture RTI as a master to the functional mock-up interface components. In *Computing,*

Networking and Communications (ICNC), 2013 International Conference on. IEEE, 315–320.

[2] Jens Bastian, Christop Clauß, Susann Wolf, and Peter Schneider. 2011. Master for co-simulation using FMI. In *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. Linköping University Electronic Press, 115–120.

[3] Frédéric Boussinot and Robert De Simone. 1991. The ESTEREL language. *Proc. IEEE* 79, 9 (1991), 1293–1304.

[4] David Broman, Christopher Brooks, Lev Greenberg, Edward A Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. 2013. Determinate composition of FMUs for co-simulation. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*. IEEE Press, 2.

[5] David Broman, Lev Greenberg, Edward A. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. 2014. *Requirements for Hybrid Cosimulation.* Technical Report UCB/EECS-2014-157. EECS Department, University of California, Berkeley. to appear in HSCC, Seattle, WA, April 14-16, 2015.

[6] Stefano Centomo, Julien Deantoni, and Robert De Simone. 2016. Using SystemC Cyber Models in an FMI Co-Simulation Environment. In *19th Euromicro Conference on Digital System Design 31 August - 2 September 2016 (19th Euromicro Conference on Digital System Design)*, Vol. 19. Limassol, Cyprus. https://doi.org/10.1109/DSD.2016.86

[7] Benoit Combemale, Cédric Brun, Joël Champeau, Xavier Crégut, Julien Deantoni, and Jérome Le Noir. 2016. A Tool-Supported Approach for Concurrent Execution of Heterogeneous Models. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*.

[8] Benoit Combemale, Julien Deantoni, Benoit Baudry, Robert B. France, Jean-Marc Jézéquel, and Jeff Gray. 2014. Globalizing Modeling Languages. *IEEE Computer* (June 2014), 10–13. https://hal.inria.fr/hal-00994551

[9] Fabio Cremona, Marten Lohstroh, Stavros Tripakis, Christopher Brooks, and Edward A. Lee. 2016. FIDE - An FMI Integrated Development Environment. In *Symposium on Applied Computing*.

[10] Julien Deantoni, Cédric Brun, Benoit Caillaud, Robert B. France, Gabor Karsai, Oscar Nierstrasz, and Eugene Syriani. 2015. *Domain Globalization: Using Languages to Support Technical and Social Coordination.* Springer International Publishing, Cham, 70–87. https://doi.org/10.1007/978-3-319-26172-0_5

[11] Colin Fidge. 1991. Logical time in distributed computing systems. *Computer* 24, 8 (1991), 28–33.

[12] Virginie Galtier, Stephane Vialle, Cherifa Dad, Jean-Philippe Tavella, Jean-Philippe Lam-Yee-Mui, and Gilles Plessis. 2015. FMI-based Distributed Multi-simulation with DACCOSIM. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium (DEVS '15)*. Society for Computer Simulation International, San Diego, CA, USA, 39–46.

[13] David Garlan and Mary Shaw. 1993. An introduction to software architecture. *Advances in software engineering and knowledge engineering* 1, 3.4 (1993).

[14] Abir Ben Khaled, Mongi Ben Gaid, Nicolas Pernet, and Daniel Simon. 2014. Fast multi-core co-simulation of Cyber-Physical Systems: Application to internal combustion engines. *Simulation Modelling Practice and Theory* 47 (2014), 79 – 91. https://doi.org/10.1016/j.simpat.2014.05.002

[15] Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* 21, 7 (1978), 558–565.

[16] Matias Ezequiel Vara Larsen, Julien Deantoni, Benoit Combemale, and Frédéric Mallet. 2015. A behavioral coordination operator language (BCOoL). In *Model Driven Engineering Languages and Systems (MODELS), 2015 ACM/IEEE 18th International Conference on*. IEEE, 186–195.

[17] Matias Ezequiel Vara Larsen, Julien Deantoni, Benoit Combemale, and Frédéric Mallet. 2015. A Model-Driven Based Environment for Automatic Model Coordination. In *Models 2015 demo and posters*.

[18] Edward A Lee. 2008. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*. IEEE, 363–369.

[19] Nenad Medvidovic and Richard N Taylor. 1997. A framework for classifying and comparing architecture description languages. *ACM SIGSOFT Software Engineering Notes* 22, 6 (1997), 60–76.

[20] Modelisar. 2014. FMI for Model Exchange and Co-Simulation. (July 2014). https://fmi-standard.org/downloads#version2

[21] Himanshu Neema, Jesse Gohl, Zsolt Lattmann, Janos Sztipanovits, Gabor Karsai, Sandeep Neema, Ted Bapty, John Batteh, Hubertus Tummescheit, and Chandraseka Sureshkumar. 2014. Model-based integration platform for FMI co-simulation and heterogeneous simulations of cyber-physical systems. In *Proceedings of the 10 th International Modelica Conference; Lund; Sweden*. Linköping University Electronic Press, 235–245.

[22] George A Papadopoulos and Farhad Arbab. 1998. Coordination models and languages. *Advances in computers* 46 (1998), 329–400.

[23] Vitaly Savicks, Michael Butler, and John Colley. 2014. Co-simulating Event-B and continuous models via FMI. In *Proceedings of the 2014 Summer Simulation Multiconference*. Society for Computer Simulation International, 37.

[24] Tom Schierz, Martin Arnold, and Christoph Clauß. 2012. Co-simulation with communication step size control in an FMI compatible master algorithm. In *Proceedings of the 9th International MODELICA Conference; Munich; Germany*. Linköping University Electronic Press, 205–214.

[25] Jean-Philippe Tavella, Mathieu Caujolle, Charles Tan, Gilles Plessis, Mathieu Schumann, Stéphane Vialle, Cherifa Dad, Arnaud Cuccuru, and Sébastien Revol. 2016. Toward an Hybrid Co-simulation with the FMI-CS Standard. (April 2016). https://hal-centralesupelec.archives-ouvertes.fr/hal-01301183 Research Report.

[26] S. Tripakis. 2015. Bridging the semantic gap between heterogeneous modeling formalisms and FMI. In *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. 60–69. https://doi.org/10.1109/SAMOS.2015.7363660

[27] Bert Van Acker, Joachim Denil, Hans Vangheluwe, and Paul De Meulenaere. 2015. Generation of an Optimised Master Algorithm for FMI Co-simulation. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium (DEVS '15)*. Society for Computer Simulation International, San Diego, CA, USA, 205–212.

[28] Baobing Wang and John S. Baras. 2013. HybridSim: A Modeling and Co-simulation Toolchain for Cyber-physical Systems. In *Proceedings of the 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications (DS-RT '13)*. IEEE Computer Society, Washington, DC, USA, 33–40. https://doi.org/10.1109/DS-RT.2013.12

Real-Time Capable Retargeting of Xilinx MicroBlaze Binaries using QEMU

Real-Time Capable Retargeting of Xilinx MicroBlaze Binaries using QEMU – A Feasibility Study

Irune Yarza Mikel Azkarate-askasua

IK4-Ikerlan Technology Research Centre, Dependable Embedded Systems
Pº J.M. Arizmendiarrieta, 2
20500 Arrasate-Mondragón, Spain
iyarza@ikerlan.es, emailmazkarateaskasua@ikerlan.es

Kim Grüttner
OFFIS - Institute for Information Technology
Eschwerweg 2
26121 Oldenburg, Germany
kim.gruettner@offis.de

Wolfgang Nebel
C.v.O. Universität Oldenburg
Ammerländer Heerstr. 114-118
26121 Oldenburg, Germany
nebel@informatik.uni-oldenburg.de

ABSTRACT

The great expansion and fast evolution of embedded systems market and the known advantages of the use of multi-core processors in this area are generating interest on improved embedded systems technologies (e.g., shrinking transistor size, new on-chip architectures), which at the same time are shortening the obsolescence periods of the underlying hardware. As a consequence, software designed for those platforms (a.k.a legacy code), that might be functionally correct and validated code, will be lost when changing the underlying Instruction Set Architecture (ISA) and peripherals. Given that many embedded systems execute Real-Time (RT) applications, the legacy code migration problem directly affects RT systems. Dynamic Binary Translation (DBT) techniques have been widely used for the migration of legacy code to a new hardware platform. However, there are no works which consider their use for RT legacy code migration. Therefore, this paper analyzes the suitability of a DBT based emulator, Quick EMUlator (QEMU), as a mean for RT legacy code migration, mainly focusing on its temporal capacities. To this end, a test framework has been constructed to check and compare the timing behavior of a bare-metal execution on a Xilinx MicroBlaze processor against a QEMU emulated MicroBlaze running on an ARM Cortex-A9 processor. Results show that the proposed approach could provide hard RT performance in 55% and soft RT performance in 74% of our considered benchmarks.

Keywords

Dynamic Binary Translation, legacy code, Real-Time Systems, retargeting

RAPIDO, January 22–24, 2018, Manchester, United Kingdom

DOI: https://doi.org/10.1145/3180665.3180671

1. INTRODUCTION

Next generation embedded systems must provide better performance, increased dependability, and energy efficiency, while achieving a cost effective product withing a reduced time-to-market. This transition is being encouraged by the important role of multi-core processors in the area of embedded systems, providing concurrent resources and increased performance rates at lower clock frequencies and lower power consumption [7].

The continuous demand for improved embedded system technologies and the fact that processor manufacturers are universally moving to multi-core solutions has provoked the shortening of the obsolescence period of embedded systems hardware and, as a consequence, the need to deal with legacy code. Legacy code is characterized by some particular properties: usually runs on obsolete hardware which is slow and expensive to maintain [23], has no or outdated documentation [21], and is essential for the company [5] since it comprises business knowledge [22].

Due to the fact that classical process models focus on the first stage of software life cycle (development) instead of operation or maintenance stages, the process of updating legacy systems is usually complex, error-prone, time consuming and requires high cost investments. One of the existing forms to deal with legacy code is binary translation. The aim of binary translation is to transform existing binaries (for the legacy ISA) into binaries for the new architecture, either statically or dynamically (during emulation). This way, the legacy system is transformed into a new form that will improve its operation, system capability, functionality, performance or evolvability at a lower cost, schedule or risk to the customer [19].

1.1 Motivation

Industry needs a RT software re-targeting approach that can be easily ported to different target and host processors and we see binary translation technologies as a good candidate to implement those needs. Although binary translation techniques have been successfully applied to non-RT applications, only [11] and [15] consider RT systems in their proposed approaches. When dealing with the migration process of legacy RT systems, not only the functional behavior of the software, but also the temporal behavior has to be

preserved. This is a typical pattern of a reactive RT system, found in many control applications. One example is a plant controller system, where the controller is periodically triggered, obtains new (sensor) data and updates its (actuator) outputs within a specific period of time, which has to be smaller than its triggering period.

1.2 Contributions

The overall goal of this research is to enable the migration of RT embedded legacy code to a new hardware platform with guaranteed RT performance. This paper describes the first steps of this research. To this end, this work analyzes the suitability of QEMU, as a machine-adaptable DBT tool, for its use in a RT property conserving re-targeting process. This approach is applicable when the original source-code is not available or there is a need to keep it unmodified.

The main contribution of this paper is the construction of a test environment to check if the timing behavior on the legacy architecture can be fulfilled on the new architecture. Due to the multi-host and -target quality of QEMU, our proposed test framework can easily be ported and used in other guest/host combinations. The detailed technical contributions of this paper are:

- Overview of the state of the art in static and dynamic code translation techniques heeding portability, embedded systems or RT legacy code.
- Establishment of a Xilinx MircoBlaze QEMU environment on a Xilinx Zynq ARM Cortex-A9 processing system.
- Setup of a high-resolution execution time measurement for periodically triggered software running on QEMU.
- A feasibility study for RT capable re-targeting of Xilinx MicroBlaze binaries using QEMU on a Xilinx Zynq ARM Cortex-A9 processing system.

1.3 Outline

The remainder of the paper is organized as follows. An overview of related work in the area of machine-adaptable static and dynamic code translation techniques for embedded systems is provided in Section 2. Then, Section 3 presents the proposed solution and describes the implementation details of the MircoBlaze QEMU environment on a Xilinx Zynq ARM Cortex-A9 processing system. Section 4 describes the construction of the test framework with a high-resolution execution time measurement of a periodically triggered software. Section 5 executes the feasibility study and discusses the experimental results. An outlook on future work and a conclusion is given in Section 6.

2. RELATED WORK

Binary translation techniques have been widely studied and developed in the last two decades. Some of the utilities of binary translation techniques include fast simulation of instruction sets, resource protection and management (safety), and software security enforcement. Moreover, binary translation appears to be a standard approach for legacy software migration, as the software that runs on the legacy hardware can be migrated to a new hardware platform without a considerable expense of time, effort and money.

The first binary translation techniques were developed in the late 1980s for academic researches and commercial products [10]. In 1987, HP developed one of the earliest commercial binary translation systems to migrate source HP 3000 programs to the new Precision Architecture [6]. Followed by the IBM System/370 simulator running on top of an IBM RT(RISC) PC, MIMIC [17]. Later, binary translation techniques were used by many other affiliations aiming a migration path for existing software.

Software based binary translation systems can be classified into two categories: Static Binary Translation (SBT) and DBT. The former translates the binary code offline, before the program is executed, while the later translates the binary code at runtime, during program execution.

Nowadays, most binary translation approaches adopt dynamic translation, since this technique can easily handle indirect branches and can perform optimization based on program's run-time behavior. However, as translation and optimization time counts for a part of the execution time (which incurs execution overhead), dynamic translation approaches cannot perform aggressive optimizations. On the contrary, static translation approaches can perform whole program optimization without influence on run-time overhead but they must deal with code discovery, code location and self modifying code issues. Some researches have even implemented hybrid binary translation techniques to take advantage of the strengths of both methods.

Given the great amount of binary translation systems, just those heeding portability, embedded systems and/or RT applications will be considered here.

2.1 Static Binary Translation

The TIBBIT project [11] was the first binary translation approach developed for embedded RT applications (which are not assumed to be user-level processes) that needed to be migrated to a different processor but still maintaining the externally observable timing behavior. To this end, both, the legacy application and the operating system code, used by the application, are packed in a black box that is executed on top of the host operating system. During the translation process, timing code is inserted into each translation block, to maintain the timing behavior the same as in the original processor. If the execution is running faster than it did in the old processor, other tasks are run until the execution is back on schedule.

The UQBT [9] was the first SBT tool designed with portability in mind. UQBT translates the user-level target binary into a Higher-Level Register Transfer Language (HRTL), which is a machine-independent representation, and then the HRTL is translated into host machine binary code, what makes the tool easily and inexpensively portable to new target and host architectures. To handle indirect calls that could not be discovered at static time, the UQBT uses an interpreter.

Chen et al. [8] developed a SBT solution for embedded systems. Their tool directly migrated ARM binaries to a MIPS-like architecture, without using an Intermediate Representation (IR), which enabled applying architecture specific optimization but at the same time hindered translator's retargetability. Their translation tool was able to migrate user-level ARM binaries without using a run-time emulator or DBT support.

Heinz [15] proposed a system-level SBT approach for the migration of RT legacy software. However, instead of dynamically computing a delay, as Cowsgell and Zary did on the TIBBIT project, the computation of the delay is shifted from run-time to compile-time. The translator selects from a set of precomputed delays the appropriate value according to the context of a program point, so there is no need to keep track of the execution time on the source machine.

DisIRer [16] is a multi-platform SBT tool based on the GNU Compiler Collection (GCC) compiler infrastructure. DisIRer converts user-mode target binary programs into GCC IR and then translates IR into host binary code using GCC optimizer and back end. The fact that it is based on GCC makes the translator cost effective and retargetable.

Shen et al. [18] worked out a Low Level Virtual Machine

(LLVM) based retargetable SBT tool for embedded systems, LLBT. LLBT translates ARM user-mode instructions into LLVM IRs and then LLVM IRs are translated into machine code for multiple ISAs. LLVM infrastructure provides LLBT means for optimization and retargetability. In order to make the system suitable for embedded systems, LLBT reduces the size of the address mapping table.

2.2 Dynamic Binary Translation

UQDBT [20] was the first retargetable DBT approach. It is a dynamic version of UQBT. Just as UQBT, UQDBT does not support system-level emulation. UQDBT separates the system into machine-dependent and machine-independent parts, it uses a machine independent intermediate representation (I-RTL). However, the machine adaptability of the translator comes at the cost of performance. To improve generated code, UQDBT performs generic hot path optimizations that are applicable on different types of machines.

QEMU [4] is a machine emulator, build upon a fast and portable DBT system. The initial versions of QEMU used to translate the target source code into micro-operations implemented by a small sequence of C code which was then pre-compiled into host machine code using GCC. Newer versions of QEMU use Tiny Code Generator (TCG) to translate target source code into a machine independent representation IR and then translate IRs into host machine code. In order to reduce system overhead, QEMU applies Translation Block (TB) chaining, which avoids direct jumps by directly jumping to the next TB without returning control to the execution engine.

Baiocchi et al. have proposed different approaches to adapt DBT to embedded systems with Scratchpad Memory (SPM), which is a single cycle access and low power memory. To this end, [1] proposes an approach to manage the translated code cache, which is placed on the SPM, by reducing the amount of additional code injected by the translator. This reduces the cost for re-translating application code, and avoiding eviction of frequently executed code. [3] proposes to bound the size of the translated code cache, located also on the SPM, and to reduce the amount of code injected by the translator to control the execution flow, which accounts for about the 70% of the code in the translation cache. Furthermore, [2] presents a code cache spread between SPM, for most frequently used code, and main memory, where code cache reduction techniques described in [3] are also applied.

Guha et al. also proposed techniques to adapt DBT to embedded systems, by presenting in [12] 4 different techniques to reduce the amount of code cache occupied by exit stubs, a balanced path selection policy, and a selective flushing approach, which are combined with auxiliary code optimization to improve memory efficiency and performance [13].

CrossBit [24] is a multi-source and multi-target DBT approach, which, as the rest of portable solution do, uses an independent intermediate representation known as VInst to translate user-level target source code into host machine code. CorssBit uses profiling information to determine the hot code where optimizations are applied. These optimizations are generic, so that they can be applied to any host architecture. CrossBit also applies Basic Block chaining to avoid direct jumps for further reducing system overhead.

2.3 Summary

Our approach aims to provide a migration path for (soft) RT legacy code. However, the proposed solution is based on DBT techniques, which have never before been applied on the migration process of RT legacy code. The only two approaches (TIBBIT [11] and Heinz [15]) that aim at the migration of RT legacy code make use of SBT techniques. Moreover, unlike existing RT code migration approaches, the proposed solution is machine-adaptable, so it can easily be ported to other source or target architectures. In addition, our approach ports an embedded bare-metal application from a legacy target to a new host architecture, providing a system-level emulation approach. Table 1 provides a summary of the related work section and compares our approach the other discussed solutions.

3. RT LEGACY CODE MIGRATION ARCHITECTURE

In order to provide a migration path for soft RT legacy applications, QEMU as a portable and low overhead DBT tool has been used. QEMU runs on top of a pre-build minimalistic Ubuntu distribution for Zynq, available on Xilinx's Wiki. However, the kernel has been configured using the PREEMPT_RT patch. The host processor is a ARM Cortex-A9 and the target processor (legacy processor emulated on QEMU) is a MicroBlaze.

The ported legacy code block is treated as a black box that is being reused without any knowledge of its implementation. The only observations of the functional behavior and timing properties of this block box are done through external monitoring of its Input/Output (I/O) ports and variables. This is a typical pattern of a reactive RT system, found in many control applications. As shown in Figure 1, the application is periodically triggered (at t_0, t_1, ...), obtains new data from the sensors and after a period of time (dt) updates the actuators. For an appropriate behavior of the system, the duration of the application (dt) must be bellow the execution period ($t_{n+1} - t_n$). When moving to the emulated architecture, this timing behavior still has to be fulfilled. In other words, since the aim of this approach is to establish an emulation framework rather than a simulation framework, the measured end-to-end duration of the application when running on the emulated architecture must be below the execution period.

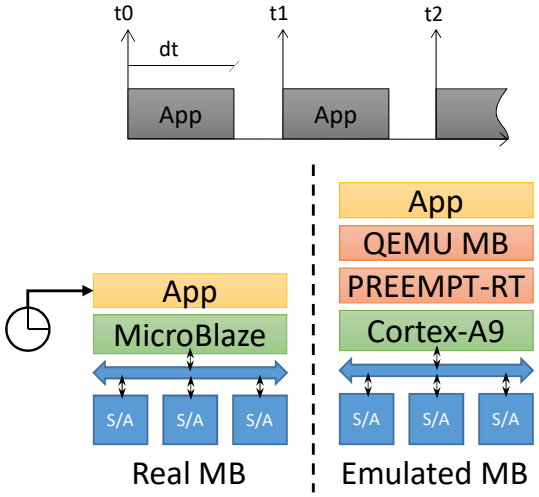


Figure 1: RT legacy code migration path. On the left hand the reactive RT application running on the legacy hardware. On the right hand the same application running on the new hardware platform using the portability layer (QEMU MB + PREEMPT_RT). The top side of the figure shows the timing behavior that must be fulfilled on both environments.

Table 1: Related work summary

Name	Static/Dynamic	Machine-adaptable	RT legacy Code	Embedded Systems	User-/System-level
TIBBIT [11]	Static	-	X	X	System-level
UQBT [9]	Static	X	-	-	User-level
Chen [8]	Static	-	-	X	User-level
Heinz [15]	Static	-	X	X	System-level
DisIRer [16]	Static	X	-	-	User-level
LLBT [18]	Static	X	-	X	User-level
UQDBT [20]	Dynamic	X	-	-	User-level
QEMU [4]	Dynamic	X	-	X	System- and User-level
Baiocchi [1–3]	Dynamic	X	-	X	No info.
Guha [12, 13]	Dynamic	X	-	X	No info.
CrossBit [24]	Dynamic	X	-	-	User-level
Our Approach	Dynamic	X	X	X	User- and System-level

3.1 Linux PREEMPT_RT

In order to fulfill the previously mentioned timing requirements, the PREEMPT_RT patch has been applied on the Linux kernel. This patch aims to improve the kernel's scheduler latency and response time, thus achieving a more deterministic Linux environment without the need of a specific Application Programming Interface (API). The PREEMPT_RT patch has been applied to Linux kernel version 4.9, which is configured to use both CPUs in SMP mode. The QEMU task has been declared as a RT task with highest priority, as shown in Listing 1.

```
#include <sched.h>
/* We use 49 as PREEMPT_RT uses 50 for the
   priority of the kernel task sets and
   interrupt handler by default. */
#define MY_PRIORITY (49)

int main(int argc, char **argv) {
  struct sched_param param;
  /* Declare ourselves as a real-time task */
  param.sched_priority = MY_PRIORITY;
  if (sched_setscheduler(0, SCHED_FIFO,
                         & param) == -1) {
     perror("sched_setscheduler failed");
     exit(-1);
  }
}
```

Listing 1: Adapt QEMU to PREEMPT_RT (vl.c)

3.2 Launching QEMU-MicroBlaze

QEMU was specially designed to emulate Linux guest systems, which is reflected in a special start-up procedure. Through the `-kernel` argument, a binary file (usually the Linux kernel itself) is passed to the system, which is loaded on the Operating System (OS) starting memory address position (0x90000000 for the MicroBlaze). Therefore, to run a bare metal application, the MicroBlaze local memory has to be placed at address position 0x90000000, which is configured using Vivado address editor. Moreover, when compiling the benchmarks, the linker script needs to be modified to make sure the program is placed at the correct memory location.

The QEMU source code, as provided in the QEMU GIT repository only allows the emulation of a MicroBlaze on the Spartan-3A-DSP-1800 board. For this reason, it was necessary to provide QEMU information about our target system. This is done using a Device Tree Blob (DTB) file, which can be easily obtained from the Device Tree Source (DTS) files generated by the Xilinx Software Development Kit (XSDK) design tool. However, the DTS files are not properly generated, since XSDK does not allocate the memory, and defines the machine as a 64-bit system. Listing 2 and listing 3 shows the changes made in the generated DTS files.

```
DDR2SDRAM: memory@90000000 {
        device_type = "memory";
        reg = <0x90000000 0x10000000>;
};
```

Listing 2: Allocate memory on the MicroBlaze (`system-top.dts`)

```
address_cells = <0x1>
```

Listing 3: Set system to 32 Bit (`pl.dtsi`)

4. FEASIBILITY ANALYSIS: TEST FRAMEWORK

A test framework has been constructed to perform a feasibility study of the approach described above. This framework provides means to perform a measurement based Worst Case Execution Time (WCET) analysis on both, the real and emulated MicroBlaze processors. Results obtained are then analyzed and compared in Section 5. For the execution timing measurements on the emulated MicroBlaze, a Linux high-resolution timer has been used. Whereas for execution timing measurements on the real MicroBlaze, a custom clock cycle counter has been implemented in the FPGA. Figure 2 depicts the setup for the feasibility study.

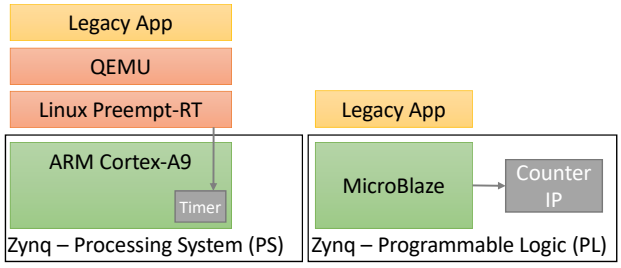


Figure 2: Test framework. On the left hand the test environment used to measure execution time on the emulated MicroBlaze (Cortex-A9). On the right hand the test environment used to measure execution time on the real MicroBlaze.

4.1 ARM Cortex-A9 (Emulated MicroBlaze)

QEMU uses TCG to translate target source code into a machine independent IR and then IR into host machine code. For QEMU

a TB is the unit of a basic block. Once translated, such TBs are stored in the code cache to avoid re-translation of code. Moreover, to avoid returning control to the emulation manager after the execution of each TB, QEMU chains consequentially executed TBs. The `tb_find` function is in charge of locating the next TB according to the target Program Counter (PC) value. If this search fails, code is generated and stored in the code cache to be used in future runs. When the code cache overflows, all translated TBs are removed.

In order to be able to do timing measurements on the emulated environment, there is a need to identify the start and end of legacy code execution. To this end, the target source code has been modified by adding a function call at the beginning and ending of the program execution. By using a function call it is ensured that there is a branch, so in the QEMU translation process this instruction will be the first in the TB. In the QEMU source code, start and end function calls are identified through the target PC value. The variables `start_pc` and `end_pc` represent the PC value of start and end function calls. The PC values of start and end function calls are dependent of the target legacy code (each application has its own values), so these values are passed through arguments (`-start-pc`, `-end-pc`) when launching QEMU. However, as already mentioned, QEMU chains consequently executed TBs in order to reduce the translation overhead. This causes the chaining of start and end TBs to former TBs, and control does not return to the emulation manager. Since there is no way to identify the start and end of execution, chaining of start and end TBs has been avoided as shown in Listing 4.

```
1  if (!tb->invalid &&
2      tb->pc != (0x90000000 + start_pc) &&
3      tb->pc != (0x90000000 + end_pc)) {
4          tb_add_jump (last_tb, tb_exit, tb);
5  }
```

Listing 4: Avoid chaining start/end TBs (`cpu-exec.c`)

Since start and end TBs are never chained, it is possible to identify them. As shown in Listing 5, function `tb_find`, which is in charge of finding the next TB to be executed, `start_pc` and `end_pc` are identified. When start and end TBs are identified, the `clock_gettime()` POSIX interface is used to access high-resolution timers (running at nanosecond time granularity) and get the start and end timing information. Then, the `diff` function is used to get the elapsed time.

```
1  #include <time.h>
2
3  struct timespec diff(struct timespec start,
4                       struct timespec end) {
5    struct timespec temp;
6    if ((end.tv_nsec - start.tv_nsec) < 0) {
7      temp.tv_sec = end.tv_sec - start.tv_sec-1;
8      temp.tv_nsec = 1000000000 +
9                     end.tv_nsec - start.tv_nsec;
10   } else {
11     temp.tv_sec = end.tv_sec - start.tv_sec;
12     temp.tv_nsec = end.tv_nsec - start.tv_nsec;
13   }
14   return temp;
15 }
16
17 static inline TranslationBlock *tb_find(
18   CPUState *cpu, TranslationBlock *last_tb,
19   int tb_exit) {
20   [...]
21   cpu_get_tb_cpu_state(env, &pc, &cs_base, &flags);
22   if (pc== (0x90000000 + start_pc)) {
23     clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &time1);
24   }
25   else if (pc == (0x90000000 + end_pc)) {
26     clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &time2);
27     if (runs < program_runs) {
28       temp[runs] = (diff(time1, time2).tv_nsec +
29                     (diff(time1, time2).tv_sec *
30                      1000000000));
31       runs++;
32     }
33   }
34   [...]
35 }
```

Listing 5: Identify benchmark start/end (`cpu-exec.c`)

4.2 MicroBlaze

In order to get execution time measurements on the real MicroBlaze, as shown on Figure 3, a clock cycle counter has been implemented as a custom hardware block with an AXI slave interface. This counter Intellectual Property (IP) block counts the clock cycles elapsed from enabling to disabling and stores the obtained result inside a Block-RAM (BRAM). The BRAM is also an AXI slave, so the MicroBlaze can read data from it. The MicroBlaze has also an AXI slave General Purpose Input/Output (GPIO), which is internally connected to the counter IP block to provide the enabling/disabling capability to the benchmark running on top of the MicroBlaze.

The counter IP block is in charge of storing the measurement values. When the GPIO is enabled, the IP block starts counting and stops when the GPIO is disabled. Then, the IP block stores the measurement in the BRAM and waits until the next execution starts. Since the BRAM size is set to 64KB, the benchmarks were executed 16383 times. This is the maximum amount of data that can be stored, given that each measurement value has a size of 4 bytes.

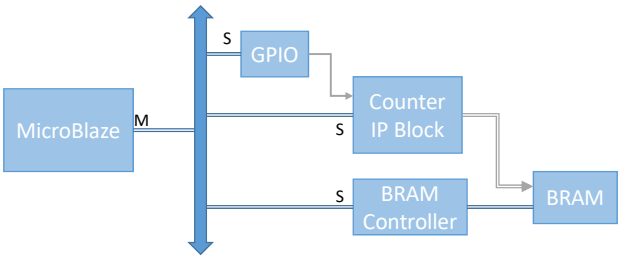


Figure 3: Configuration used to measure execution time on the real MicroBlaze. A custom counter IP block is used and measurement values are stored in a BRAM.

5. FEASIBILITY RESULTS

For the feasibility study, a selection of 27 WCET benchmark programs, provided by the Mälardalen WCET research group [14], have been applied by comparing the execution times of these benchmarks between the real and emulated MicroBlaze. Since these benchmarks contain great variety of algorithms (including loops, nested loops, use of array and/or matrixes and use of floating point operations), we get a wide analysis of the timing behavior of the proposed solution.

5.1 Platform configuration

To evaluate the proposed solution, the ZC702 evaluation board with a Zynq-7000 XC7Z020 SoC has been used. The Zynq-7000 provides software and hardware programmability, integrating a Dual-core ARM Cortex-A9 (Processing System - PS) and an FPGA (Programmable Logic - PL) on a single chip.

For the execution time analysis on the real MicroBlaze, a bitstream has been generated for the Zynq Programmable Logic (PL)

part. The bitstream includes the MicroBlaze version 10.0 configured with: local BRAM memory for instruction and data, 64KB each (the use of cache is disabled); without a floating point unit; and optimization set to performance (uses a five-stage pipeline). Moreover, it includes the peripherals described on subsection 4.2.

The same configuration has been used to generate the DTB file for launching the QEMU MicroBlaze. However, on the emulated MicroBlaze, apart from the already mentioned peripherals, an AXI Timer and an AXI Interrupt Controller are needed in order to be able to generate the DTS files. The boot image, First Stage Boot Loader (FSBL) binary and u-boot binary on the Ubuntu distribution have been replaced by the boot.bin file generated on XSDK using the FSBL binary and the u-boot binary for our specific Vivado project.

The target source code that runs on the real and emulated MicroBlaze has been compiled whit the same MicroBlaze GCC (mb-gcc) without any optimization (-O0).

5.2 Evaluation process and results

To get the results, every benchmark has been executed 16383 times on the real and emulated MicroBlaze (without relaunching QEMU, so that we take advantage of translated code caching) and timing data has been collected. Together with the executed WCET benchmarks, an empty application has also been analyzed. Running the empty application provided a way to measure the timing overhead introduced by the underlying system, composed of QEMU and Linux PREEMPT_RT.

In order to solve scaling problems, result have been clustered into 5 different graphs. These graphs show a comparison between the emulated and real execution time average value and 98%-quantile. The standard deviation is represented as an error bar on the average value. When analyzing the results, benchmarks should be classified into two groups according to the ratio between computation complexity and control flow statements. We have selected benchmarks which are mainly composed of conditional statements and contain simple computations (e.g., array search, binary search, bubblesort, compress, duff, faculty, janne complex, statemate, switch, ud), and benchmarks with complex computations and few control flow statements (e.g., crc, dctint, dhrystone, edn, fft, insertsort, lms, ludcmp, matmult, ndes, petrinet, prime, qsort, qurt, select, sqrt, statistics). The former group is expected to run faster on the real MicroBlaze, since control flow statements hinder TB chaining on QEMU and simple computations do not take advantage of the new and more powerful host architecture. Whereas the latter group, is expected to run faster on the emulated MicroBlaze, because the host architecture can handle better complex computations and the absence (or low amount) of control flow statements supports efficient TB chaining.

As shown in Figure 4, four of the analyzed benchmarks (binary search, bubblesort, insertsort and switch) have shorter execution times when running on the real MicroBlaze (avg real) than the timing overhead of the emulation environment itself (empty benchmark avg emulated). According to the empty benchmark, the emulation environment introduces a timing offset of almost 10us. Therefore, it is impossible that these benchmarks run faster on the emulated MicroBlaze than on the real one.

Figures 5, 6, 7 and 8 show the timing results of the benchmarks that are above the empty application emulation threshold. Further analysis should be done on the timing behavior of array search, dctint and fft benchmarks. Despite containing complex computations in their algorithms, dctint and fft benchmarks run faster on the real MicroBlaze than on the emulated one. The main reason behind this behavior might reside on the intensive cache use of

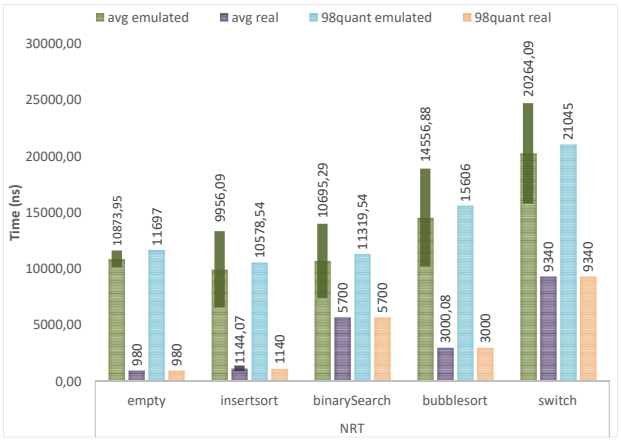


Figure 4: Timing results of benchmarks below the empty application emulation threshold. These benchmarks can not overtake the timing offset introduced by the emulation environment.

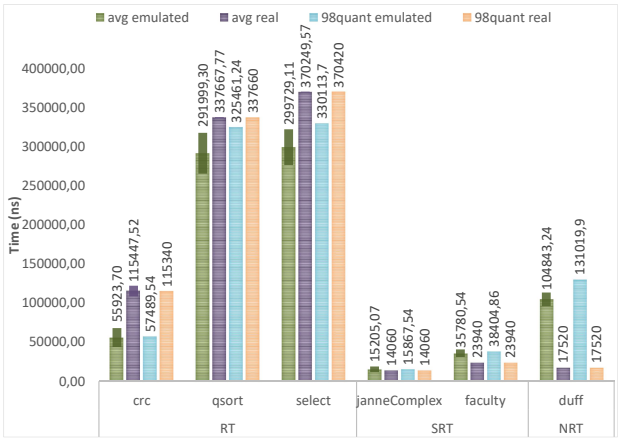


Figure 5: Timing results of benchmarks above the empty application emulation threshold (Cluster 1). The proposed solution could provide hard RT performance for crc, qsort and select benchmarks and soft RT performance for janneComplex and faculty.

these benchmarks and the more frequent cache misses on the ARM platform. Even though we expected array search would run faster on the real MicroBlaze, since it contains simple computations and many conditional statements, it runs faster on the emulated MicroBlaze. This could be, because the nested loops on the benchmark can be resolved by QEMU TB chaining infrastructure.

According to the results, the proposed solution could provide hard RT performance for 55% of the benchmarks. These are the benchmarks marked as RT in the figures, which run faster on the emulated hardware than on the real one. Whereas, soft RT performance could be provided for 74% of the benchmarks. This value corresponds to the benchmarks marked as SRT, which run almost as fast on the emulated hardware as on the real MicroBlaze. Obviously, the 74% value contains the RT and SRT benchmarks. Thus, the non RT benchmarks represent 26%.

6. CONCLUSIONS AND FUTURE WORK

The current research work aimed to study the feasibility of a dynamic code translation for RT legacy binaries. In our feasibility study, QEMU has been adapted to run a Xilinx MicroBlaze emulator on a Xilinx Zynq ARM processor and has been modified to per-

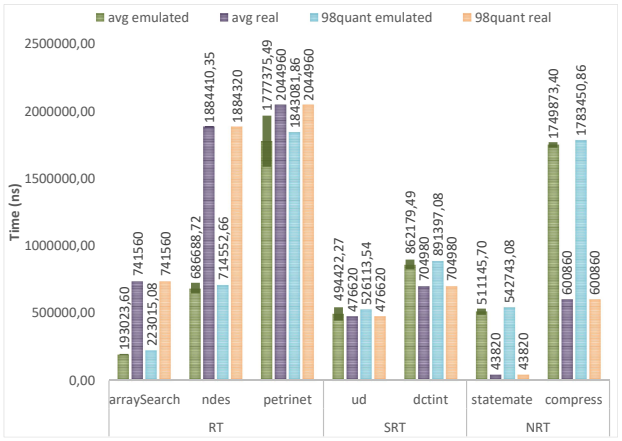


Figure 6: Timing results of benchmarks above the empty application emulation threshold (Cluster 2). The proposed solution could provide hard RT performance for arraySearch, ndes and petrinet benchmarks and soft RT performance for ud and dctint.

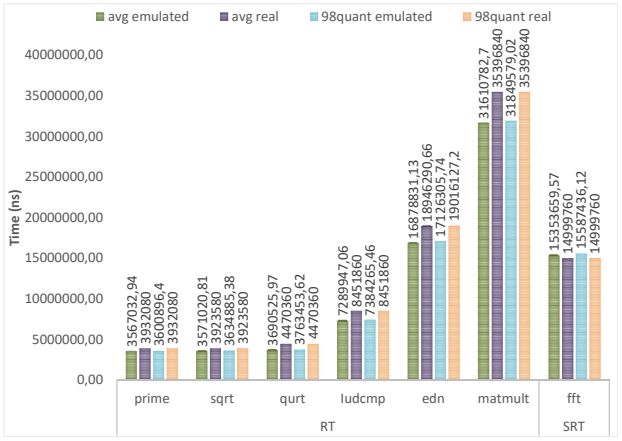


Figure 7: Timing results of benchmarks above the empty application emulation threshold (Cluster 3). The proposed solution could provide hard RT performance for prime, sqrt, qurt, ludcmp, edn and matmult benchmarks and soft RT performance for fft.

form timing measurements. Afterward, a WCET benchmark suite has been executed on the emulated target and a bare-metal MicroBlaze target. From the experimental result analysis, we conclude that QEMU could be a feasible path for pure functional soft RT legacy code block migration. It is a machine-adaptable solution and supports full system emulation, moreover new embedded devices can easily be added to the system.

As already mentioned in the introduction, this work described early results. Future work considers analysis of the caching behavior inside QEMU (TB cache tracing) and on the Linux platform through available performance counters. This will enable a much better observability and interpretation of the influence of the QEMU and Linux caching effect on the execution time of emulated applications. Furthermore, I/O and interrupt virtualization (already supported by QEMU) will be considered to establish the required periodic schedule for the class of targeted reactive real-time systems in conjunction with a more realistic industrial case-study. Last but not least, the integration of multiple emulated processors, running together on a multi-core host processor, shall be addressed. Such a complex integration scenario could then be further extended

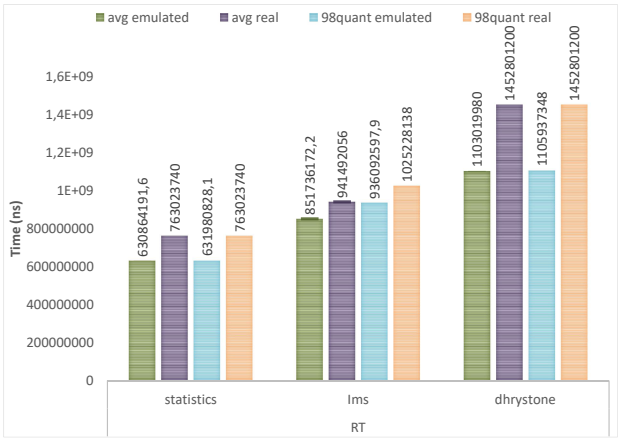


Figure 8: Timing results of benchmarks above the empty application emulation threshold (Cluster 4). The proposed solution could provide hard RT performance for statistics, lms and dhrystone benchmarks.

into a mixed emulated legacy and non-emulated non-legacy code integration under consideration of (soft) real-time constraints.

Acknowledgment

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687902 (SAFEPOWER).

The authors would like to thank Xabier Ormaetxea and Philipp Ittershagen for their support on the research described on this publication.

References

[1] J. Baiocchi, B. R. Childers, J. W. Davidson, J. D. Hiser, and J. Misurda. Fragment cache management for dynamic binary translators in embedded systems with scratchpad. In *Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 75–84. ACM, 2007.

[2] J. A. Baiocchi and B. R. Childers. Heterogeneous code cache: using scratchpad and main memory in dynamic binary translators. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pages 744–749. IEEE, 2009.

[3] J. A. Baiocchi, B. R. Childers, J. W. Davidson, and J. D. Hiser. Reducing pressure in bounded dbt code caches. In *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, pages 109–118. ACM, 2008.

[4] F. Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[5] K. Bennett. Legacy systems: Coping with success. *IEEE software*, 12(1):19–23, 1995.

[6] A. B. Bergh, K. Keilman, D. J. Magenheimer, and J. A. Miller. Hp-3000 emulation on hp precision architecture computers. *Hewlett-Packard Journal*, 38(11):87–89, 1987.

[7] G. Blake, R. G. Dreslinski, and T. Mudge. A survey of multicore processors. *IEEE Signal Processing Magazine*, 26(6):26–37, 2009.

[8] J.-Y. Chen, W. Yang, T.-H. Hung, H.-M. Su, and W.-C. Hsu. A static binary translator for efficient migration of arm-based applications. In *Workshop on Optimizations for DSP and Embedded Systems*. Citeseer, 2008.

[9] C. Cifuentes and M. V. Emmerik. Uqbt: adaptable binary translation at low cost. *Computer*, 33(3):60–66, 2000.

[10] C. Cifuentes and V. Malhotra. Binary translation: static, dynamic, retargetable? In *1996 Proceedings of International Conference on Software Maintenance*, pages 340–349, 1996.

[11] B. Cogswell and Z. Segall. Timing insensitive binary to binary translation of real time systems. In *Workshop on Architectures for Real-Time Applications, ISCA*, 1995.

[12] A. Guha, K. Hazelwood, and M. L. Soffa. Reducing exit stub memory consumption in code caches. In *International Conference on High-Performance Embedded Architectures and Compilers*, pages 87–101. Springer, 2007.

[13] A. Guha, K. Hazelwood, and M. L. Soffa. Memory optimization of dynamic binary translators for embedded systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, 9(3):22, 2012.

[14] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The Mälardalen WCET benchmarks – past, present and future. In B. Lisper, editor, *WCET2010*, pages 137–147, Brussels, Belgium, July 2010. OCG.

[15] T. Heinz. Preserving temporal behaviour of legacy real-time software across static binary translation. In *Proceedings of the 1st workshop on Isolation and integration in embedded systems*, pages 1–4. ACM, 2008.

[16] Y.-S. Hwang, T.-Y. Lin, and R.-G. Chang. Disirer: Converting a retargetable compiler into a multiplatform binary translator. *ACM Transactions on Architecture and Code Optimization (TACO)*, 7(4):18, 2010.

[17] C. May. *Mimic: a fast system/370 simulator*, volume 22. ACM, 1987.

[18] B.-Y. Shen, J.-Y. Chen, W.-C. Hsu, and W. Yang. Llbt: an llvm-based static binary translator. In *Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems*, pages 51–60. ACM, 2012.

[19] S. R. Tilley and D. Smith. Perspectives on legacy system reengineering, 1995.

[20] D. Ung and C. Cifuentes. Machine-adaptable dynamic binary translation. In *ACM SIGPLAN Notices*, volume 35, pages 41–51. ACM, 2000.

[21] C. Wagner and C. Wagner. *Model-Driven Software Migration*. Springer, 2014.

[22] M. Wahler, R. Eidenbenz, C. Franke, and Y. A. Pignolet. Migrating legacy control software to multi-core hardware. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 458–466, 2015.

[23] B. Wu, D. Lawless, J. Bisbal, J. Grimson, V. Wade, D. OŠSullivan, and R. Richardson. Legacy system migration: A legacy data migration engine. In *Proceedings of the 17th International Database Conference (DATASEM'97)*, pages 129–138, 1997.

[24] Y. Yang, H. Guan, E. Zhu, H. Yang, and B. Liu. Crossbit: a multi-sources and multi-targets dbt, 2010.

XXXII

Design Space Pruning and Computational Workload Splitting for Autotuning OpenCL Applications

Design Space Pruning and Computational Workload Splitting for Autotuning OpenCL Applications

Ahmet Erdem Davide Gadioli

Gianluca Palermo Cristina Silvano

Department of Electronics, Information and Bioengineering
Politecnico di Milano
e-mail: name.surname@polimi.it

ABSTRACT

Recently, OpenCL standard reached much wider audiences due to the increasing number of devices supporting it. At the same time, we have observed an increase of differences among devices that support OpenCL. This situation offers to developers, who want to get high performance, a large spectrum of platforms. Given the additional OpenCL platform parameters alongside application specific parameters, the design space for exploration is seriously large. Furthermore, availability of more than one kind of device allows distribution of computation on the heterogeneous platform. Automatic design space exploration frameworks are one of the recent approaches to address these problems and to reduce the burden of programmers. In this work, we present our automatic and efficient technique to prune the design space before moving on to the exploration phase and we propose a new method for splitting the computational tasks to computing devices on heterogeneous platforms.[1]

1. INTRODUCTION

The recent advances in computer architecture made heterogeneous computer systems available to not only data centers and supercomputers, but also to commercial personal computers. Especially, with the advent of AMD APUs and Intel CPUs which include integrated GPUs, the heterogeneity of modern machines has increased. Furthermore, enabling discrete GPUs for general purpose computing has added another type of computation device to the system. While each system has provided different granularity of parallelism which needs to be properly exploited, the communication between various computation devices must also be handled according to the requirements of application as well.

[1]This work is partially funded by the EU H2020 FET-HPC program under grant 671623 ANTAREX.

RAPIDO, January 22–24, 2018, Manchester, United Kingdom

DOI: https://doi.org/10.1145/3180665.3180669

Open Computing Language (OpenCL), maintained by the Khronos consortium [4], is an open standard for developing parallel applications on heterogeneous systems by abstracting the underlying compute machine. OpenCL adopts data parallel approach by describing the parallel computations as a group of *work-items*, named *work-groups*. In this hierarchical mechanism, a kernel function is executed in parallel by each work-item in the work-group. A kernel function describes how each work-item defines the operations carried out on a single data. Therefore, the collection of work-items under all work-groups together expresses the data parallelism for an application. Although OpenCL guarantees that the execution of the application is portable between the devices conforming the OpenCL standard, it does not guarantee the performance to be optimal. Especially, moving applications to different types of architectures like from CPU to GPU may result significant loss of performance. This is the reason why OpenCL is not considered performance portable. Heterogeneous platforms performance portability represents a challenging research issue.

One naive solution to performance portability is to develop separate kernel functions for each device the application is supposed to run. This solution makes development of application dramatically complicated when the system is heterogeneous, because of explicit management of multiple command queues and contexts in the presence of multiple vendors on the system. Moreover, this approach has more design flaws:

- The application developer must have knowledge of all the device's architectures.
- The application must be manually split between existing devices on the platform.
- The developer should have access to all the devices in order to test and profile on them.
- The number of devices targeted by the application is limited, consequently future architectures are impossible to target in a performance optimal way without modifying the application code.

The performance portability problem of OpenCL applications has been approached either by tuning significant parameters as described in [6] or by introducing Domain-specific languages to annotate kernel, to generate more specialized OpenCL code[2].

From another perspective, it is not always possible to access these parameters to tune if they are not being exposed by developers. The work of [1] tackles this problem by coalescing *work-groups* using compiler transformations while preserving the correctness of application.

In Glinda framework presented by [8], a specific application with possible imbalanced workload is analyzed and used as a case study for their load balancing and autotuning framework.

Sharing similar vision with [8], Alok Prakash et al. demonstrate in their work [7] how they approached the problem of utilization of heterogeneous computing platform on an embedded device.

The open source library named Maestro [9], which is introduced by Kyle Spafford et al., employs autotuning techniques to find optimal work-group size and load balancing between multiple devices on heterogeneous platforms.

Similarly, in the work of [3], adaptive ways of selecting faster architecture using source-to-source polyhedral compiler explored. Concurrent runs on devices are not used to accelerate the execution, but rather to use the fastest device that finishes the execution.

In this work, we introduce an automation of extraction of OpenCL platform parameters and usage of the information that are gathered to aid the tuning process. Furthermore, we propose a new method to split the computational workload to different OpenCL devices on heterogeneous systems.

2. PROPOSED METHODOLOGY

2.1 Parameter Space Pruning

The procedure of autotuning an OpenCL application in order to get optimum performance without any concerns of the underlying architecture of the platform, requires a set of parameters that define characteristics of the machine. In the case of OpenCL, these platform specific parameters are stated by the OpenCL standard itself. Furthermore, it is possible to gather them using the querying framework which is provided by the OpenCL standard. With these information gathered, it is possible to determine the size of the exploration space and then using intelligent methods for searching optimum design space.

Besides platform parameters, there might be also application specific parameters that are tightly related to platforms capabilities. An example of this situation is the well-known tiled version of the matrix multiplication. The size of the tiles are considered as application parameters and due to nature of the algorithm there is a sharing of information between work-items on the elements of the same tile. Due to OpenCL architecture design, this kind of communication requires local memory to be used. Therefore, tile size is directly related to local memory usage which is a limited resource of the platforms.

There are some problems related to searching for the optimum configuration, design space is larger for even simple applications. For instance, Nvidia Fermi architecture has limitations on work-group sizes for each dimension allowing up to 1024 work-items for first and second data-dimensions, while 64 work-items for the third dimension, resulting in 2^{26} different configurations. Most of these configurations are not feasible (e.g. the total number of work-items may exceed devices capabilities), in the sense that the kernel may not even launch or may fail during execution, due to unfeasible configurations parameters. Moreover these failed attempts of kernel launches do not provide any information about the sample that has been taken from design space. Hence, effort and time are wasted on these unfeasible configurations.

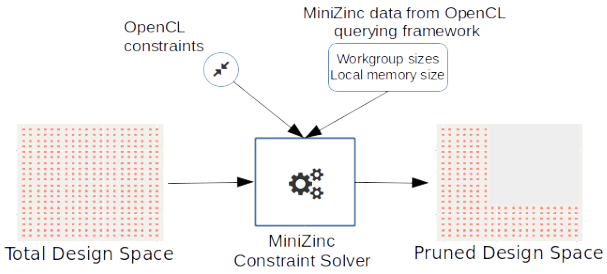


Figure 1: Design Space Pruning

To address this issue, the work in [6] presented a design space exploration flow that includes constraint programming to prune the design space and eliminate unfeasible solutions. This helps the reduction of the design space. Moreover, it only uses configurations which make sense within the scope of OpenCL standard. Fig. 1 demonstrates this idea. Constraint Solver shown in Figure 1, eliminates the samples which do not comply with the constraints from the design space. And as output of the solver, a pruned design space is generated such that all configuration samples are compliant with the constraints.

Our work aims at improving the pruning phase by automating the extraction of platform specifications, to find constraints that are valid for all the OpenCL devices in the target system. Therefore, application programmer only needs to insert constraints related to application itself. For constraint programming, we use the MiniZinc [5] constraint modelling language. Using *clGetDeviceInfo* function provided by OpenCL querying framework, for each OpenCL compliant device available on the machine we generate MiniZinc data files which include the following information about the device:

- maximum *work-group* size for each dimension.
- maximum total number of work-groups considering all dimensions.
- number of compute units on the device.
- local memory size of the device.

In addition to these device parameters, a set of constraints that can be deduced from the rules defined by the standard [4], has been used to generate platform constraint model using MiniZinc constraint programming language. Thus, together with application constraints provided by programmer, it is possible to prune the design space effectively. The generated platform constraint model contains the following rules:

- The total number of *work-groups* launched must be less than or equal to maximum *work-group* size.

$$workgroup_x * workgroup_y * workgroup_z <= max\_total\_wg \quad (1)$$

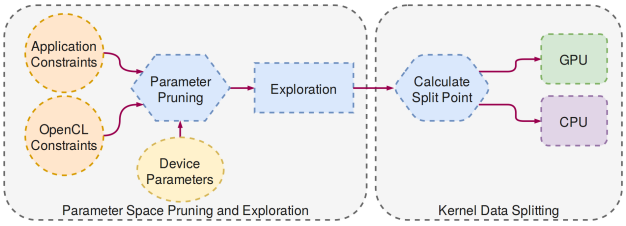


Figure 2: The proposed methodology

- Each global work-item dimensions must be multiple of corresponding *work-group* dimension size.

$$\begin{aligned} global_x \% workgroup_x == 0 \\ global_y \% workgroup_y == 0 \\ global_z \% workgroup_z == 0 \end{aligned} \quad (2)$$

- The total number of work-groups should be equal or greater than number of compute units. Otherwise, there will be idle compute units.

$$\begin{aligned} global_x/workgroup_x + global_y/workgroup_y + \\ global_z/workgroup_z >= num\_compute\_units \end{aligned} \quad (3)$$

Using both constraints coming from platform specifications and application domain, the total design space will be reduced to a collection of configurations that satisfy these constraints. This will reduce the exploration of the space, which is crucial for the next phase.

2.2 Computational Workload Splitting

Heterogeneous computing architectures are widely adopted, thus being able to utilize this heterogeneity is crucial for achieving higher performance systems. Therefore, in this work we propose a new method that splits huge OpenCL computing kernels into smaller chunks. Then it maps those partial computations to different devices in order to make use of all the existing OpenCL devices.

Since the development of OpenCL framework has been inspired from GPUs, which have been conceived for data-parallel computing, we are focusing on data-parallel computations by using the *NDRange* functionality of OpenCL framework. When using *NDRange*, it is possible to launch kernels on an iteration space where each iteration processes one element of a set of data. The goal is to be able to prune that iteration space and distribute the portions of it to the available devices in a efficient manner.

To achieve this goal, we should address the following issues:

- It is required the performance knowledge of a kernel on each device of the heterogeneous platform in order to find the right split point.
- After splitting, the chosen work-group sizes may become ill-advised for kernel execution.

In order to find a balanced splitting point, we needed performance information of the target kernel on each device. To acquire this information, we measured execution time of the remaining configurations after parameter space pruning described in Section 2.1. After the exploration, the candidate

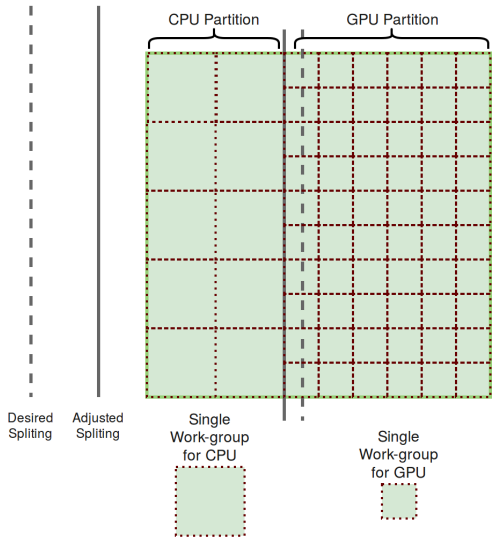


Figure 3: Global work space splitting

configurations with the lowest execution time for each device are chosen for splitting decision. In Figure 2, the proposed methodology has been shown to give high level perspective.

First, execution times are converted to speed values for each device by taking multiplicative inverse. Then, these speed values are used for calculating *split factors* for each device using Equation 4; where $split\_factor_i$ is the *split factor* of i^{th} device and N is the number of devices.

$$split\_factor_i = exe\_speed_i / (\sum_{j=0}^{N-1} exe\_speed_j) \quad (4)$$
$$\text{where} \quad i = 0, \ldots, N-1s$$

Since split factors define how much computation will take place in each OpenCL device, the sum of all the split factors must be equal to 1.0. Otherwise, there would be residue computation that is missing from the output of the task. It is fairly easy to observe that $\sum_{i=0}^{N-1} split\_factor_i = 1.0$ is satisfied by substituting $split\_factor_i$ by the Equation 4.

While split factors are calculated from performance measurements to give good hints about how to load balance between different devices in heterogeneous environments, dividing the global work sizes using a generic split factors, may easily result in partitions with fractional work sizes which are invalid for the OpenCL standard.

For instance, let us consider a global work size of 512 work-items and split factors of 0.4 and 0.6 for the two devices. In this case, the share of device 1 would be 204.8 work-items while device 2 has a share of 307.2 work-items. Because the concept of work-items is intrinsically indivisible, the splitting factors are impractical.

Moreover, it is important to keep the optimum work-group sizes found by exploration for each device, since the performance is strongly related to work-group sizes. This situation is problematic because different devices usually have different optimal work-group sizes. In [7], a single work-group size has been used for both GPU and CPU. In contrast, this work introduces a new technique to overcome this limitation by adjusting the splitting factors minimally while taking into

account the optimal work-group sizes discovered in the exploration phase.

Equation 5 defines the global work size. where wg_i is the work-group size and γ_i is the number of work-groups for i^{th} device. For simplicity, it only considers one dimension of the iteration space. Moreover, it can be extended for the multi-dimensional case, since splitting operation can be applied to each dimension separately.

$$G = \sum_{i=0}^{N-1} wg_i * \gamma_i \tag{5}$$

The number of work-groups that are needed for each device is calculated using devices' optimal work-group sizes and splitting factor S_i (Eq. 6).

$$\gamma_i = \frac{G * S_i}{wg_i} \tag{6}$$

It is important to note that at this stage, the numbers of work-groups γ_i are real values, hence OpenCL framework will not launch successfully. To deal with this issue, the number of work-groups are reduced to an integer number for all devices (Eq. 7).

$$\hat{\gamma_i} = \lfloor \gamma_i \rfloor \tag{7}$$

With the integer number of work-groups $\hat{\gamma_i}$, the splitting factors are recalculated (Eq. 8a) in order to find the residue (Eq. 8b).

$$\hat{S_i} = \frac{wg_i * \hat{\gamma_i}}{G} \tag{8a}$$

$$S_R = \sum_{i=0}^{N-1} S_i - \sum_{j=0}^{N-1} \hat{S_j} \tag{8b}$$

Having a leftover part of the computation ($G * S_R$), we determine how much work each device needs to process, in order to compute residue (Eq. 9a and Eq. 9b). Then we choose the device that requires the minimum amount of work to cover the remaining computations.

$$\tilde{\gamma_i} = \lceil \frac{G * S_R}{wg_i} \rceil \tag{9a}$$

$$\tilde{S_i} = \frac{wg_i * \tilde{\gamma_i}}{G}, \tag{9b}$$

Using this method, it is possible to preserve the desired work-group sizes while adjusting split factor minimally. We achieve this by reducing the number of work-groups conservatively per device, then choosing the appropriate device for the residue work to be computed on. An example of splitting and partitioning of global work space into work-groups is illustrated in Figure 3.

An interesting feature of this method is that, when work-group sizes of devices are not multiple of each other, there will be overlap between the partitions of data that are computed on devices. However the redundant computations will be minimum due to choice of the device which requires least amount of work for residue. In case the work-group sizes are multiple of each other, there will be no overlap at all. This outcome is observed because when work-group sizes are multiple of each other, larger work-groups can perfectly be accommodated by smaller work-groups.

Algorithm 1 Calculates the best theoretical performance

```
function TheoreticalHeteroPerf(splitFactors, exeTimes)
    devicePerfs ← empty list
    i ← 0
    while i < number of devices present do
        devicePerfs[i] ← splitFactors[i] * exeTimes[i]
    return MAX(devicePerfs)
```

Figure 4 shows the details of the splitting phase. In this phase, we also measure how well the methodology stands against theoretically best-case performance. After calculation the best split location, the theoretical heterogeneous performance is calculated using Algorithm 1. The difference between the calculated performance and the actual heterogeneous performance is the overhead of our methodology.

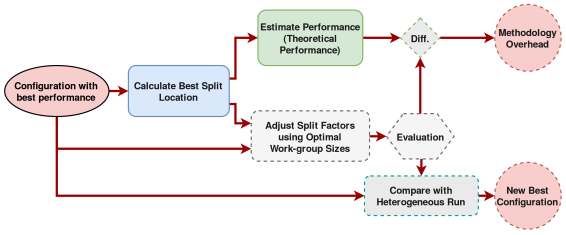


Figure 4: Proposed splitting phase in detailed

3. EXPERIMENTAL SETUP

In order to test our approach, we used two platforms. Platform 1(PLT1) consists of an Intel i7-4770 quad-core at 3.4Ghz and Intel HD Graphics 4600 with 20 Execution Units. Platform 2(PLT2) has an Intel i7-2630QM quad-core CPU at 2.0Ghz and a Nvidia GeForce GT 550M which is a mobile GPU with 96 CUDA cores.

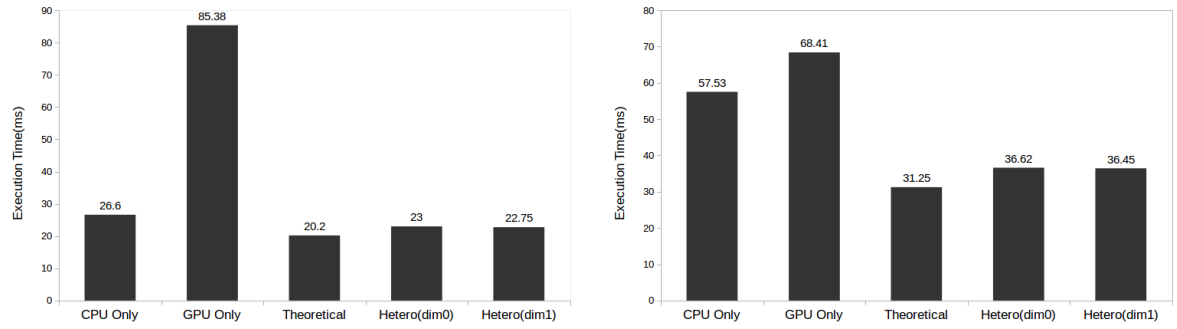
In order to validate our methodology, we have chosen two application case studies; one dimensional convolution and matrix multiplication. Both implementations make use of local memory provided by OpenCL framework as cache.

Matrix multiplication is implemented as tiled version and the size of the considered tiles is an application parameter and it is used to calculate the local memory consumption of the OpenCL kernel (Eq. 10). As a constraint to the exploration space, we fixed the workgroup sizes equal to the tile sizes.

$$local\_mem = 2 \times tile\_size^2 \tag{10}$$

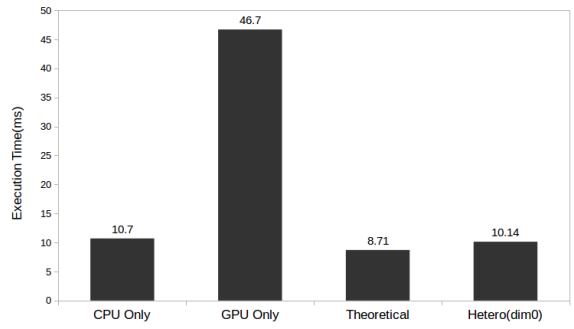
OpenCL version of the convolution operation has been implemented in a fashion that work-items, in the same work-group, share as much data as possible. This is possible by using local memory to load all the neighbouring data required for the whole work-group. Therefore, the number of work-items, as well as the mask size defined in the convolution operation, affects the local memory consumption according to (Eq. 11).

$$local\_mem = workgroup\_size + mask\_size - 1 \tag{11}$$

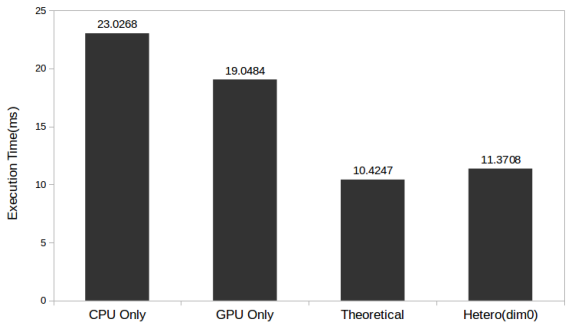


(a) MatrixMult PLT1

(b) MatrixMult PLT2



(c) Convolution PLT1



(d) Convolution PLT2

Figure 5: Execution times of heterogeneous runs

4. EXPERIMENTAL RESULTS

In this section, we present how much the design space is reduced by using our methodology. The amount of acceleration achieved due to utilization of both CPU and GPU as well as the limitations of the techniques that are introduced is examined. In order to get repeatable outcomes, we evaluated all the configurations 10 times and took the mean of the best five results for all the experiments.

4.1 Design Space Pruning

Without our pruning phase implementation, the required amount of kernel runs for the explorations are shown in Table 1. The matrix multiplication used as test case is a 1024x1024 multiplication, while the convolution operation has a mask size 625 over 2^{16} elements.

The numbers in Table 1, are generated considering only the dimensions that are used by the kernel. Matrix multiplication has a 2 dimensional iteration space, while convolution is 1 dimensional in this case. This is the reason behind the huge difference of the exploration size of the two applications. In Table 2, the number of feasible configurations that require evaluation, are dramatically reduced. The amount of reduction of the space allows us to explore all the remaining configurations.

4.2 Computation Workload Splitting

Figure 5 shows the theoretical value, execution time of the heterogeneous evaluations, along with only CPU and only GPU configurations. For the matrix multiplication case two different heterogeneous experiments are shown as dim0 and dim1 in Figure 5. It is important to notice that on PLT1 the GPU is an integrated GPU and on PLT2 the GPU is a mobile variant. Moreover, both CPUs and GPUs use the same kernel with parameters tuned to the specific device. For all the execution types, we show the configuration with the minimum execution time among the explored ones.

We may observe that the performance differences between the CPU and the GPU on PLT1 lead to less than 15% improvement for the two cases. Even the theoretical value does not suggest much better speed up (Fig. 5a, Fig. 5c). Contrarily, on PLT2 with matrix multiplication and convolution applications, there is a 54% and a 68% speed up respectively (Fig. 5b, Fig. 5d).

We have also investigated the effect of kernel data splitting on different dimensions of the iteration space. In order

Table 1: Exploration space size without pruning

	PLT1		PLT2	
	CPU	GPU	CPU	GPU
Matrix Mult.	2^{20}	2^{18}	2^{20}	2^{20}
Convolution	8192	512	8192	1024

Table 2: Exploration space size after pruning

	PLT1		PLT2	
	CPU	GPU	CPU	GPU
Matrix Mult.	7	5	7	6
Convolution	24	17	24	19

Table 3: Data split factors (CPU,GPU)

	PLT1		PLT2	
	Adjusted	Optimal	Adjusted	Optimal
MatMul.	(0.762, 0.238)	(0.75, 0.25)	(0.55, 0.45)	(0.5, 0.5)
Conv.	(0.814, 0.186)	(0.812, 0.188)	(0.45, 0.55)	(0.45, 0.55)

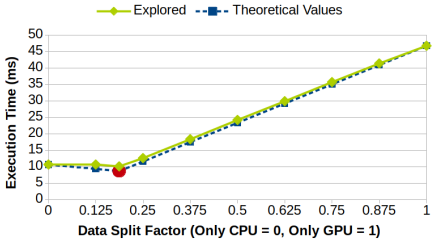


Figure 6: Split Factor Evaluations(Convolution, PLT1)

to test it, splitting technique has been applied to both dimensions (dim0, dim1) of matrix multiplication. According to the experiment results, there is no significant improvements for this case study due to the symmetric distribution of the workload.

Table 3 shows the calculated optimal split factors and split factors that are adjusted according to optimal work-group sizes explored. For each case, the first value is the computational share of CPU and second one is the share of GPU. It is important to note that for all cases, it sums up to 1.0, meaning that the computation covers all the iterations space. Overall, the difference between the optimal split point and the split point adjusted by our methodology is smaller in convolution application, due to the fact that the size of the work-groups are much smaller compared to global work size. This situation provides much a finer granularity when adjusting the splitting point, therefore leading to closer to optimal splitting. In contrast to this, the matrix multiplication work-group sizes are not as insignificant to the global work size as convolution, although differences between the adjusted and the optimal are still limited.

In order to further evaluate our methodology, we tried different split points to compare with our split point calculation (Fig. 6, Fig. 7). The splitting factor, indicated with a circle, is the factor computed by our method. On Platform 2 it is obvious that our splitting factor is the balancing point between the two devices. While it is also the case on Platform 1, the shape of the graph indicates the performance gap between two computing devices used on the platform. Additionally, the difference between the calculated execution times (dashed blue line) and the explored execution times (solid green line) is the overhead of the technique we introduced. This overhead includes adjustment of splitting point and runtime management of multiple OpenCL devices.

5. CONCLUSION & FUTURE WORK

The contribution of this work is twofold. The first contribution is a more automatic way of collecting and using OpenCL parameters as an improvement on [6]. Secondly, a new technique on how to split data between OpenCL devices while respecting work-group sizes has been introduced. Compared to [7], it is not required to have the same work-

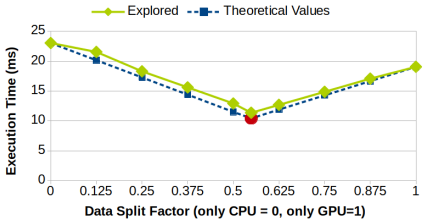


Figure 7: Split Factor Evaluations(Convolution, PLT2)

group sizes for all the devices. Removing this limitation enables the usage of better suited work-group sizes for each device.

Experimental results have shown that it is possible to significantly reduce exploration space and moreover by utilizing all the devices available on the heterogeneous platform, our methodology improves the performance.

6. REFERENCES

[1] G. Agosta, A. Barenghi, G. Pelosi, and M. Scandale. Towards transparently tackling functionality and performance issues across different opencl platforms. In *In proceedings of the Second International Symposium on Computing and Networking - Across Practical Development and Theoretical Research (CANDAR 2014)*, Dec. 2014.

[2] N. Chaimov, B. Norris, and A. Malony. Toward multi-target autotuning for accelerators. In *Parallel and Distributed Systems (ICPADS), 2014*, pages 534 – 541.

[3] J.-F. Dollinger and V. Loechner. Adaptive runtime selection for gpu. In *Proceedings of the 2013 42Nd International Conference on Parallel Processing*, ICPP '13, pages 70–79, Washington, DC, USA, 2013. IEEE Computer Society.

[4] Khronos Group. The open standard for parallel programming of heterogeneous systems. [Online; Accessed: Nov. 2015].

[5] MiniZinc. Medium-level constraint modelling language minizinc. [Online; Accessed: Dec. 2015].

[6] E. Paone, F. Robino, G. Palermo, V. Zaccaria, I. Sander, and C. Silvano. Customization of OpenCL applications for efficient task mapping under heterogeneous platform constraints. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 736–741, 2015.

[7] A. Prakash, S. Wang, A. E. Irimiea, and T. Mitra. Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 208–215. IEEE, 2015.

[8] J. Shen, A. L. Varbanescu, H. Sips, M. Arntzen, and D. G. Simons. Glinda: a framework for accelerating imbalanced applications on heterogeneous platforms. In *Proceedings of the ACM International Conference on Computing Frontiers*, page 14. ACM, 2013.

[9] K. Spafford, J. Meredith, and J. Vetter. *Maestro: Data Orchestration and Tuning for OpenCL Devices*, pages 275–286. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

Compile-Time Silent Store Elimination for Energy Efficiency : an Analytic Evaluation for Non-Volatile Cache Memory

Compile-Time Silent-Store Elimination for Energy Efficiency: an Analytic Evaluation for Non-Volatile Cache Memory

Rabab Bouziane
Univ Rennes, Inria, CNRS, IRISA
Campus de Beaulieu, 35042
Rennes Cedex, France
first.last@inria.fr

Erven Rohou
Univ Rennes, Inria, CNRS, IRISA
Campus de Beaulieu, 35042
Rennes Cedex, France
first.last@inria.fr

Abdoulaye Gamatié
CNRS, LIRMM, Univ. Montpllier
191 rue Ada, 34095
Montpellier,France
first.last@lirmm.fr

ABSTRACT

Energy-efficiency has become very critical in modern high-performance and embedded systems. In on-chip systems, memory consumes an important part of energy. Emerging non-volatile memory (NVM) technologies, such as Spin-Transfer Torque RAM (STT-RAM), offer power saving opportunities, while they suffer from high write latency.

In this paper, we propose a fast evaluation of NVM integration at cache level, together with a compile-time approach for mitigating the penalty incurred by the high write latency of STT-RAM. We implement a code optimization in LLVM for reducing so-called *silent stores*, i.e., store instruction instances that write to memory values that were already present there. This makes our optimization portable over any architecture supporting LLVM. Then, we assess the possible benefit of such an optimization on the Rodinia benchmark suite through an analytic approach based on parameters extracted from the literature devoted to NVMs. This makes it possible to rapidly analyze the impact of NVMs on memory energy consumption. Reported results show up to 42 % energy gain when considering STT-RAM caches.

Keywords

Silent stores, LLVM compiler, energy-efficiency, non volatile memory, embedded systems

1. INTRODUCTION

Memory system plays a very important role in performance and power consumption of computing devices. Previous work already pointed out that the energy related to the cache hierarchy in a system can reach up to 40% of the overall energy budget of corresponding chip [8]. As technology scales down, the leakage power in CMOS technology of the widely used SRAM cache memory increases, which degrades the system energy-efficiency. Existing memory system management techniques offer various ways to reduce the related power consumption. For instance, a technology such as SDRAM has the ability to switch to lower power modes upon a given inactivity threshold is reached. Further approaches, applied to embedded systems, deal with memory organization and optimization [23] [2].

With the increasing concern about energy consumption in both embedded and high-performance systems, emerging non-volatile memory (NVM) technologies, such as Phase-Change RAM (PCRAM), Spin-Transfer Torque RAM (STT-RAM) and Resistive RAM (RRAM), have gained high attention as they open new power saving opportunities [20]. Indeed, their very low leakage power, makes them good candidates for energy-efficiency improvement of computer systems. NVMs have variable performance, energy and endurance properties. For instance, PCRAM and RRAM have limited endurance compared to usual SRAM and DRAM technologies. STT-RAM has an endurance property that is close to that of SRAM, making it an attractive candidate for cache level integration. Nevertheless, a main limitation of NVMs at cache level is their high write latency compared to SRAM. This can be penalizing, especially for write-intensive workloads, as it leads to performance degradation, with a possible increase in global energy consumption despite the low leakage power.

In this paper, we investigate an effective usage of STT-RAM in cache memory such that its inherent overhead in write latency can be mitigated for better energy-efficiency. For this purpose, we revisit the so-called *silent store elimination* [14], devoted to system performance improvement by eliminating redundant memory writes. We target STT-RAM because it is considered as more mature than similar emerging NVM technologies. Test chips with STT-RAM already exist [10, 21] and show reasonable performance at device level compared to SRAM. More generally, the silent store elimination presented here can be applied to all NVMs for addressing their asymmetric access latencies. It may be less beneficial for technologies with less asymmetric access latencies, e.g., SRAM. An instance (or occurrence) of a store instruction is said to be silent if it writes to memory the value that is already present at this location, i.e., it does not modify the state of the memory. A given store instruction may be silent on some instances and not silent on others. The *silentness* percentage of a store instruction therefore characterizes the ratio between its silent and non-silent in-

RAPIDO, January 22–24, 2018, Manchester, United Kingdom

 ISBN 978-1-4503-6417-1/18/01…15.00
DOI: https://doi.org/10.1145/3180665.3180666

stances: high silentness therefore means a larger number of silent store instances.

While silent store elimination has been often developed in hardware, here we rather adopt a software approach through an implementation in the LLVM compiler [12]. A high advantage is that the optimization becomes portable for free, to any execution platform supporting LLVM: a program is optimized once, and run on any execution platform while avoiding silent stores. This is not the case of the hardware-level implementation. Thanks to this flexible implementation, we evaluate the profitability of silent store elimination for NVM integration in cache memory. In particular, we show that energy-efficiency improvements highly depend on the the silentness percentage in programs, and on the energy consumption ratio of read/write operations of NVM technologies. We validate our approach by exploring this tradeoff on the Rodinia benchmark suite [4]. Up to 42 % gain in energy is reported for some applications. The described validation approach is quite fast and relies on an analytic evaluation considering typical NVM parameters extracted from the literature.

In the rest of this paper, Section 2 discusses some related work. Then, Section 3 describes the general principle and implementation of our silent store elimination approach at compile-time. Section 4 applies this approach to the Rodinia benchmark suite for evaluating possible gains in energy on different applications. Finally, Section 5 gives concluding remarks and perspectives to the current work.

2. RELATED WORK

There are a number of studies devoted to energy-efficiency of NVM-based caches. Smullen et al. [24] investigated an approach that focuses on technology level to redesign STT-RAM memory cells. They lower the data retention time in STT-RAM, which induces the reduction of the write current on such a memory. This enables in turn to decrease the high dynamic energy and latency of writes.

Sun et al. [25] proposed a hybrid L2 cache consisting of MRAM and SRAM, and employed migration based policy to mitigate the drawbacks of MRAM. The idea is to keep as many write intensive data in the SRAM part as possible to reduce the number of write operations to the STT-RAM part. Hu et al. [9] targeted embedded chip multiprocessors with scratchpad memory (SPM) and non volatile main memory. They exploited data migration and re-computation in SPM so as to reduce the number of writes on main memory. Zhou et al. in [31] proposed a circuit-level technique, called Early Write Termination in order to reduce write energy. The basic idea is to terminate earlier a write transaction whenever detected as redundant.

Migration-based techniques require additional reads and writes for data movement, which penalizes the performance and energy efficiency of STT-RAM based hybrid cache. Li et al. [18] addressed this issue through a compilation method called migration-aware code motion. Data access patterns are changed in memory blocks so as to minimize the overhead of migrations. The same authors [16] also proposed a migration-aware compilation for STT-RAM based hybrid cache in embedded systems, by re-arranging data layout to reduce the number of migrations. They showed that the reduction of migration overheads improves energy efficiency and performance.

We also promote a *compile-time* optimization by leveraging redundant writes elimination on memory. While this is particularly attractive for NVMs, a few existing works already addressed the more general question about code redundancy for program optimization. In [29], the REDSPY profiler is proposed to pinpoint and quantify redundant operations in program executions. It identifies both temporal and spatial value locality and is able to identify floating-point values that are approximately the same. The data-triggered threads (DTT) programming model [26] offers another approach. Unlike threads in usual parallel programming models, DTT threads are initiated when a memory location is changed. So, computations are executed only when the data associated with their threads are modified. The authors showed that a complex code can exploit DTT to improve its performance. In [27], they proposed a pure software approach of DTT including a specific execution model. The initial implementation required significant hardware support, which prevented applications from taking advantages of the programming model. The authors built a compiler prototype and runtime libraries, which take C/C++ programs annotated with DTT extensions, as inputs. Finally, they proposed a dedicated compiler framework [28] that automatically generates data-triggered threads from C/C++ programs, without requiring any modification to the source code.

While the REDSPY tool and the DTT programming model are worth-mentioning, their adoption in our approach has some limitations: the former is a profiling tool that provides the user with the positions of redundant computations for possible optimization, while the latter requires a specific programming model to benefit from the provided code redundancy elimination (note that the DTT compiler approach built with LLVM 2.9 is no longer available, and is not compatible with the latest versions of LLVM). Here, we target a compile-time optimization in LLVM, i.e., silent store elimination (introduced at hardware level by [14]), which applies independently from any specific programming model. This optimization increases the benefits NVMs as much as possible, by mitigating their drawbacks.

3. SILENT-STORE ELIMINATION

3.1 General principle

Silent stores have been initially proposed and studied by Lepak et al. [13]. They suggested new techniques for aligning cache coherence protocols and microarchitectural store handling techniques to exploit the value locality of stores. Their studies showed that eliminating silent stores helps to improve uniprocessor speedup and reduce multiprocessor data bus traffic. The initial implementation was devised in hardware, and different mechanisms for store squashing have been proposed. Methods devoted to removing silent stores are meant to improve the system performance by reducing the number of write-backs. Bell et al. [1] affirmed that frequently occurring stores are highly likely to be silent. They introduced the notion of critical silent stores and show that all of the avoidable cache write-back can be removed by removing a subset of silent stores that are critical.

In our study, the silent store elimination technique is leveraged at compiler-level for reducing the energy consumption of systems with STT-RAM caches. This favors portability and requires no change to the hardware. We remind that this technique is not dedicated only to STT-RAM but to all

NVMs. Here, STT-RAM is considered due to its advanced maturity and performance compared to other NVM technologies. Our approach concretely consists in modifying the code by inserting *silentness* verification before stores that are identified as likely silent. As illustrated in Figure 1, the verification includes the following instructions:

1. a load instruction at the address of the store;
2. a comparison instruction, to compare the to-be-written value with the already stored value;
3. a conditional branch instruction to skip the store if needed.

(a) original code	`store @x = val`

`load y = @x`
`cmp val, y`
`bEQ next`
`store @x, val`
`next:`
	`load y = @x`
	`cmp val, y`
	`strne @x, val`

(b) transformed code	(c) with predication

Figure 1: Silent store elimination: (a) original code stores `val` at address of `x`; (b) transformed code first loads the value at address of `x` and compares it with the value to be written, if equal, the branch instruction skips the store execution; (c) when the instruction set supports predication, the branch can be avoided and the store made conditional.

3.2 Some microarchitectural and compilation considerations

While reducing the cost of cache access, the new instructions introduced in the above transformation may also incur some performance overhead. Nevertheless, specific (micro)architectural features of considered execution platforms play an important role in mitigating this penalty.

Superscalar and out-of-order (OoO) execution capabilities are now present in embedded processors. For example, the ARM Cortex-A7 is a (partial) dual-issue processor. In many cases, despite the availability of hardware resources, such processors are not able to fully exploit the parallelism because of data dependencies, therefore leaving empty *execution slots*, i.e., wasted hardware resources. When the instructions added by our optimization are able to fit in these unexploited slots, they do not degrade the performance. Their execution can be scheduled earlier by the compiler/hardware so as to maximize instruction overlap. The resulting code then executes in the same number of cycles as the original one. This instruction rescheduling is typical in OoO cores.

On the other hand, instruction predication, e.g., supported by ARM cores, is another helpful mechanism. The execution of predicated instructions is controlled via a condition flag that is set by a predecessor instruction. Whenever the condition is false, the effect of the predicated instructions is simply canceled, i.e., no performance penalty. Figure 1 (c) shows how this helps save an instruction.

At compiler optimization levels, newly introduced instruction may cause two phenomena.

1. Since the silentness-checking code requires an additional register to hold the value to be checked, there is a risk to increase the register pressure beyond the number of available registers. Additional spill-code could negatively impact the performance of the optimized code. We observed that this sometimes happens in benchmarked applications. This is easily mitigated by either assessing the register pressure before applying the silent-store transformation; or by deciding to revert the transformation when the register allocation fails to allocate all values in registers.
2. It can happen that the load we introduce is redundant because there already was a load at the same address before and the compiler can prove the value has not changed in-between. In this favorable case, our load is automatically eliminated by further optimization, resulting in additional benefits. We observed in our benchmarks that this situation is actually rather frequent.

3.3 Analysis of profitability threshold

Since our approach includes a verification phase consisting of an extra load (memory read) and compare. This overhead may be penalizing if the store is not silent often enough. Therefore, we use pre-optimization process to identify the silent stores, and especially the "promising" silent stores in the light of the profitability threshold. Hence, the compilation framework consists of two steps as follows:

1. silent store profiling, based on memory profiling, collects information on all store operations to identify the silent ones;
2. apply the optimization pass on the stores that are silent often enough.

From the data cache viewpoint (considered in isolation), the silent store optimization transforms a write into a read, possibly followed by a write. The write must occur when the store happens to not be silent. In the most profitable case, we replace a write by a read, which is beneficial due to the asymmetry of STT-RAM. On the contrary, a never-silent store results in write being replaced by a read and a write. Thus, the profitability threshold depends on the actual costs of memory accesses. In terms of energy cost, we want:

$$\alpha_{read} + (1 - P_{silent}) \times \alpha_{write} \leq \alpha_{write}$$

where α_X denotes the cost of operation X and P_{silent} is the probability of this store to be silent. This is equivalent to:

$$P_{silent} \geq \frac{\alpha_{read}}{\alpha_{write}}$$

Relative costs vary significantly depending on the underlying memory technology. Table 1 reports some values from literature. In our survey, the ratio in energy consumption between α_{write} and α_{read} varies from 1.02× to 75×.

3.4 Implementation

Our compilation process is a middle-end framework. We focused on the compiler intermediate representation (IR) shown in Figure 2, where we implemented the silent stores optimization. While this figure describes a generic decomposition into basic steps, its instantiation in our case only contains two LLVM "passes", as illustrated in Figure 3.

Through the first step, we get information about all store instructions that are present in a program (note that the

Source	NVM parameters	Ratio
Wu et al. [30] Li et al. [16], [17]	Technology: 45 nm Read: 0.4 nJ Write: 2.3 nJ	5.75
Li et al. [15]	Technology: 32 nm Read: 174 pJ Write: 316 pJ	1.8
Cheng et al. [5]	Technology: not mentioned High retention Read: 0.083 nJ Write: 0.958 nJ Low retention Read: 0.035 nJ Write: 0.187 nJ	11.5, 5.3
Li et al. [19]	Technology: 45 nm Read: 0.043 nJ Write: 3.21 nJ	75
Jog et al. [11]	Technology: not mentioned Retention time = 10 years Read: 1.035 nJ Write: 1.066 nJ Retention time = 1 s Read: 1.015 nJ Write: 1.036 nJ Retention time = 10 ms Read: 1.002 nJ Write: 1.028 nJ	1.03, 1.02, 1.025
Pan et al. [22]	Technology : 32 nm Read: 0.01 pJ/bit Write: 0.31 pJ/bit	31

Table 1: Relative energy cost of write/read in literature

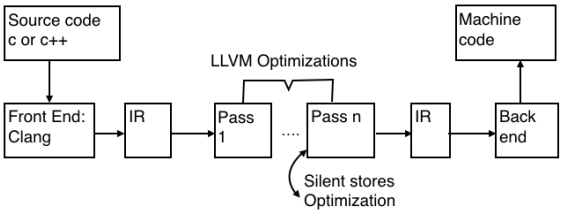


Figure 2: Implementation of the optimization in LLVM

current version of the optimization handles stores on integer and floating-point data). For that, we insert new instructions in the IR in front of every store to check whether the stored valued is equal to the already stored value. If it is the case, we increment a counter related to this particular store. Once the first pass is done, we run the application on representative inputs to obtain a summary reporting how many times the stores have been executed and how many times they have been silent. We also save their positions in the program so that we can identify them in the next pass. The output of the first pass is a file that contains all the characteristics of a store as described if Figure 3.

In a second step, where we apply the main part of the optimization: we compile again the source code, taking into account the profiling data. For each store instruction whose silentness is greater than a predefined threshold, we insert verification code, as described in Section 3.1 and illustrated in Figure 1.

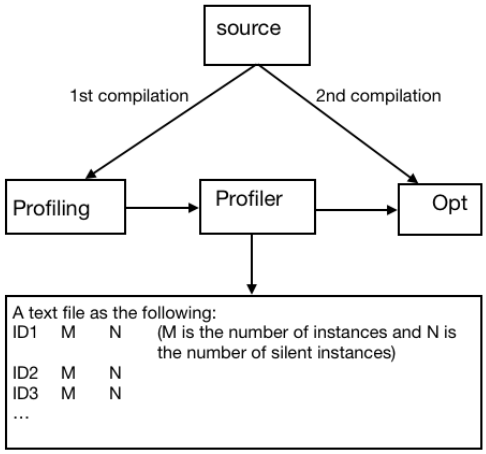


Figure 3: Design of the compilation framework

By working at the IR level we achieve three goals: 1) the source code of the application remains unmodified; 2) we do not need to be concerned by the details of the target processor and instruction set: the compiler back-end and code generator will select the best instruction patterns to implement the code; and 3) our newly introduced instructions may be optimized by the compiler, depending on the surrounding code.

4. EVALUATION ON BENCHMARKS

In their seminal 2001 paper [14], Lepak and Lipasti studied the SPEC95 benchmark suite in which high silentness percentages have been exposed. For instance, `vortex` and `m88ksim` reach respectively 64 % and 68 % of silent stores overall on PowerPC architecture (there was no per-store characterization in that paper). Such silentness levels could typically benefit from our optimization.

In this section, we present a similar analysis, uncovering silent-stores in applications, and analytically assessing the impact of our proposal, based on their characteristics. We study some applications from Rodinia benchmark [4], cross-compiled for ARM[1] and we execute them on a single core. Rodinia is composed of applications and kernels from various computing domains such as bioinformatics, image processing, data mining, medical imaging and physics simulation. In addition, it provides simple compute-intensive kernels such as LU decomposition and graph traversal. Rodinia targets performance benchmarking of heterogeneous systems.

4.1 Distribution of silent stores

The impact of eliminating a given store depends on two factors: (1) its *silentness*, i.e. how often this particular store is silent when it is executed; and (2) how many times this store is executed (obviously highly silent but infrequently executed store are not of much interest). We first study the distribution of silent-stores across applications, and then we

[1]Here, we choose ARMv7 instruction set architecture (ISA), e.g., supported by Cortex-A7 and Cortex-A15 cores, for illustration purpose. Further ISAs could be straightforwardly targeted as well, e.g., X86. This makes our code optimization portable on different processor architectures.

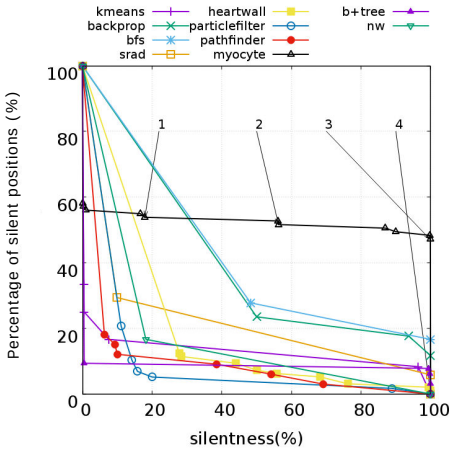


Figure 4: Percentage of silentness on the basis of the percentage of static positions in the code

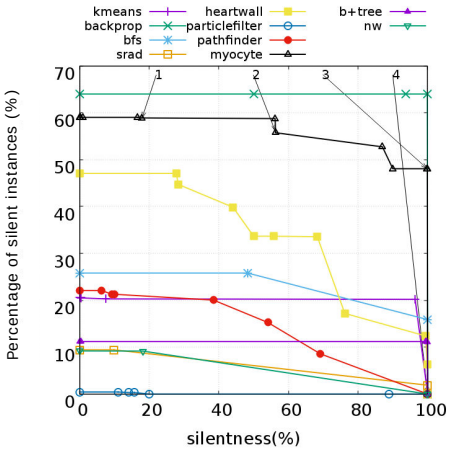


Figure 5: Percentage of silentness on the basis of the percentage of silent instances in the code

analyze the dynamic impact. A *static* distribution represents silentness through stores's positions in the code, i.e., store instructions in the assembly code file, while a *dynamic* distribution represents silentness throughout all the store instances occurring during the program execution.

Figure 4 represents the cumulative distribution of (static) silent stores in each of our applications. The plots show, for a given silentness (x-axis), what fraction of static stores achieves this level of silentness. When x increases, we are more selective on the degree of silentness, which is why all curves are decreasing. For x=100 %, we select stores that are *always* silent. This is extremely rare because, typically, at least the first instance of a store initializes a piece of data that is not already present. This explains why almost all curves reach the value y=0 when x=100 %. Conversely, when x=0, we select stores that are required to be silent at least 0 % of the time, i.e., all stores: the curves start from 100 % when x=0. The points in the curves identify the silentness of the stores in each application. For example, we observe that in the *myocyte* program, there are 53.8%, 51.6%, 47% and 0% of store instructions that are respectively 17.9%, 56.3%, 99.9% and 100% silent (see labels 1, 2, 3 and 4 in Figure 4).

In Figure 5, we take into account the weight of each store in an execution. Stores executed many times contribute more than rarely executed ones. While the x-axis still denotes the same threshold filter for silentness, the y-axis now represents the fraction of total executed stores that are silent given this threshold. For the *myocite* program, we observe that 58.8%, 56%, 48% and 0% of the silent instances are respectively 17.9%, 56.3%, 99.9% and 100% silent (see labels 1, 2, 3 and 4 in Figure 5). Also observe the case of *particlefilter* where the silentness of a number of store instructions can be high, but with a very low impact.

After studying the distribution of silent stores in a program, we can obtain an overview of the optimization benefit. If the heaviness is not significant then the gain will be marginal and even negative. In the next section, we describe how we formulate the gain based on the output of the profiling of each application.

4.2 Impact of the silent-store elimination

The impact of the silent-store elimination relies on different factors. As explained in Section 4.1, the level of silentness (high/low) and its heaviness are important. Moreover, the relative cost of read/write operation (in Joules) is critical. The ratio between the cost of a read denoted as α_R and the cost of a write denoted as α_W, can change the direction of the results. Indeed, given a ratio, the optimization outputs may vary from very bad to very good. As mentioned in Table 1, different ratios are presented in the literature. Based on that, we did a study to analyze how the impact of silent stores transformation depends drastically on the used ratio. We assume that $\alpha_W = 10$ and we vary α_R from 1 to 10 (as shown below in the formulas, only the ratio matters, hence our choice of synthetic values).

In order to formulate the gain obtained in energy after transformation, we define Δ which is the difference between the energetic cost before optimization and after optimization, denoted respectively as $cost_{base}$ and $cost_{opt}$. In other words, Δ is the expected benefit from the transformation. Since we replace a store with a read and maybe a store if the store is not always silent, then: Δ_i is defined as follows:

$$\Delta_i = cost_{base} - cost_{opt} = \alpha_W - (\alpha_R + \alpha_W \times (1 - P_i))$$

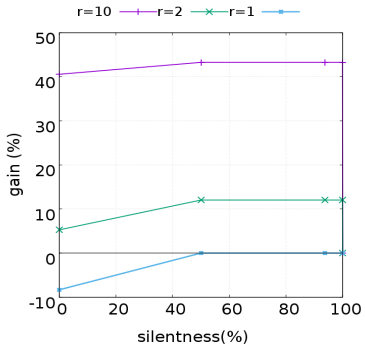
where P_i is the probability of silentness.
After transforming all the silent stores, then Δ will be:

$$\Delta = \sum_{i \in \{silent\}} (\alpha_W - (\alpha_R + \alpha_W \times (1 - P_i)))$$

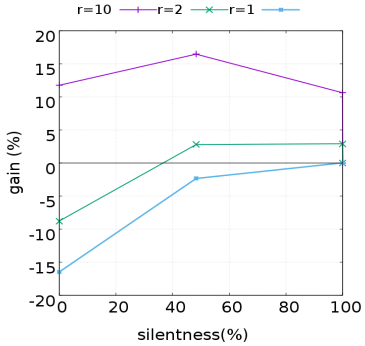
where P_i is the probability of silentness of different transformed stores. In order to get the effective gain, Δ is divided by $cost_{base}$ which is the energetic cost of all read and write operations before optimization:

$$Gain = \frac{\sum_{i \in \{silent\}} (\alpha_W - (\alpha_R + \alpha_W \times (1 - P_i)))}{\alpha_R \times N_R + \alpha_W \times N_W}$$

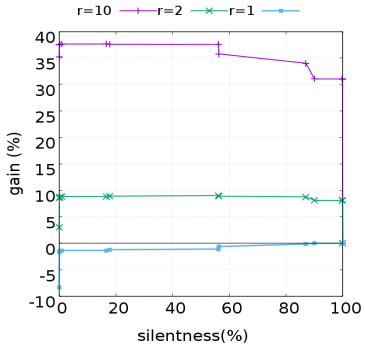
where N_R and N_W are respectively the number of load and write operations, obtained from the profiling. Considering $r = \frac{\alpha_W}{\alpha_R}$, then we obtain:



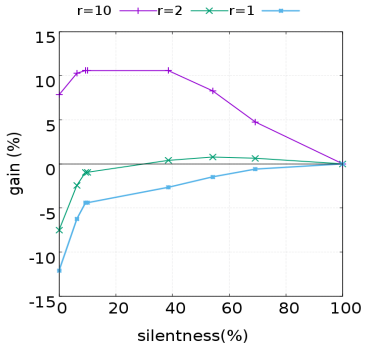
(a) backprop



(b) bfs



(c) myocyte



(d) pathfinder

Figure 6: Energy gain according to the silentness threshold of Rodinia applications, and their associated $r = \alpha_W/\alpha_R$ ratio: most sensitive applications.

$$Gain = \frac{\sum_{i\in\{silent\}}(P_i - 1/r)}{N_W/r + N_R}$$

In Figure 6 and 7, we plot this energy gain for each benchmark and for three values of the ratio r: 10, 5, and 1.For each configuration, we plot the gain obtained when optimizing silent-stores whose silentness is greater than a given value (same x-axis as in previous figures).

First, we observe that, without exception, the higher is the ratio, the higher is the gain. In other words, the more asymmetric is the non volatile memory, the more the transformation is beneficial. This is expected, and confirms the validity of our approach.

Second, a ratio $r = 1$ means symmetric memory accesses. For this technological node, our optimization cannot be beneficial. This is confirmed graphically: the gain represented by the blue curve is always negative, reaching 0 only when all stores are 100 % silent.

Generally speaking, the maximum value indicates the best threshold for the silent-store optimization. For a given non volatile memory technology, characterized by the r value, application developers and system designers can plot such curves and identify the best threshold.

The four Rodinia applications reported in Figure 6 are those which can benefit the most of silent store elimination given their silentness thresholds and considered r ratios. The remaining applications, displayed in Figure 7, only show a marginal benefit. *backprop* and *myocite* can deliver large energy gains up to 42 % when $r = 10$, while *bfs* can reach 16 %, and *pathfinder* 10 %.

In the intermediate case $r = 2$, the behavior basically depends on applications. The *bfs* program shows a negative gain with low silentness and positive gain with high silentness (from 48 %). The *srad* program shows negative gain for all silentness percentages, while the *myocyte* program shows an interesting gain through all silentness percentages (see Figure 6 and 7).

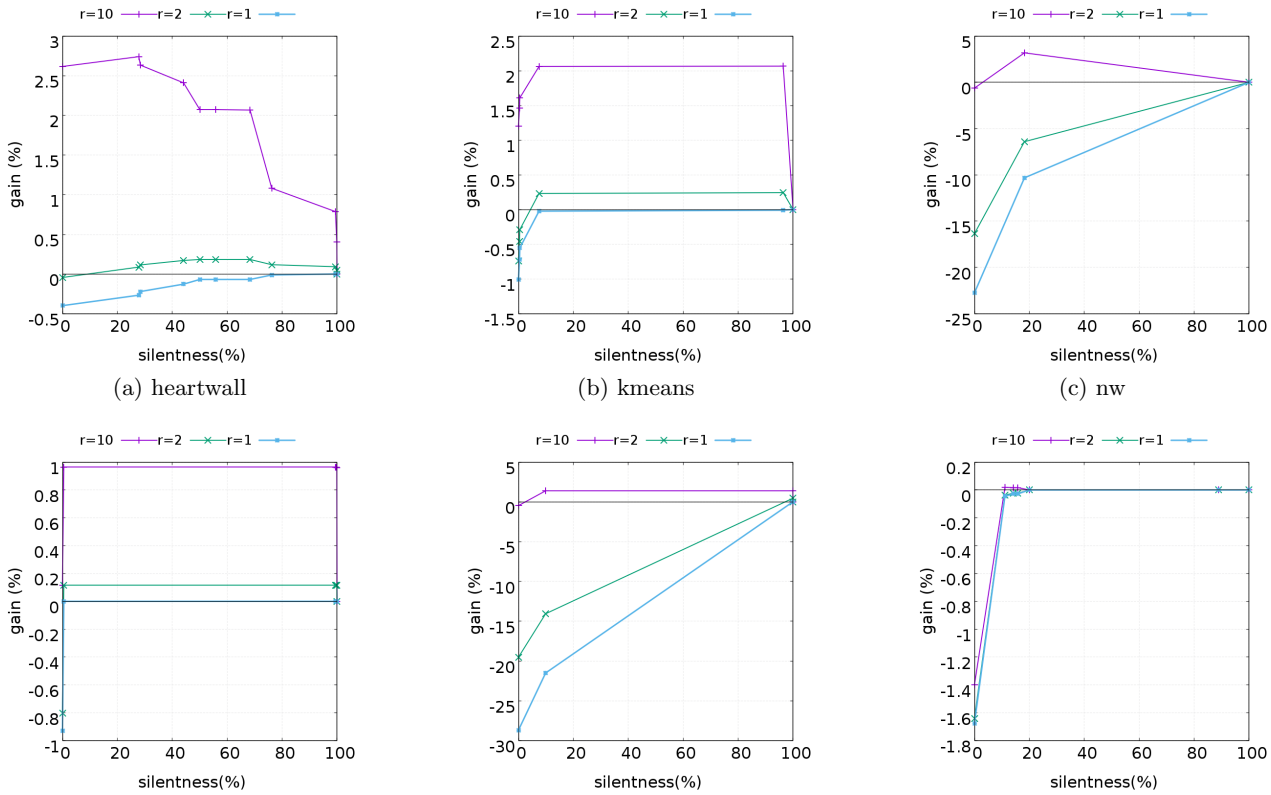
Depending on the silentness profile of the application, the gain can be fairly flat, as in *backprop*, *myocite* or *b+tree*, or vary significantly with the silentness threshold, as in *pathfinder*, or *heartwall*. In the latter case, the energy consumption critically depends on the choice of the threshold.

Finally, curves typically show an ascending then descending phase. This derives from the following phenomenon. Consider the value $x = 1$, i.e. the code is the original not optimized (except for the extremely rare case where a store is silent in exactly 100 % of the cases). When lowering x, we increase the number of store instructions that are optimized, and we increase the gain because we add highly silent stores. But when x keeps decreasing, we start adding stores that may not be silent enough and start causing degradation.

As a final note, remember that loads may be eliminated by compiler optimization. This opportunity is not captured by our above analytical model. It is hence pessimistic, and actual results should be better than our findings.

5. CONCLUSION AND PERSPECTIVES

In this paper, we presented a rapid evaluation approach for addressing the effective usage of STT-RAM emerging non volatile memory technology in cache memory. We proposed a software implementation of silent store elimination



(a) heartwall (b) kmeans (c) nw

(d) b+tree (e) srad (f) particlefilter

Figure 7: Energy gain according to the silentness threshold of Rodinia applications, and their associated $r = \alpha_W/\alpha_R$ ratio: marginally sensitive applications.

through LLVM compiler, in order to mitigate the costly write operations on STT-RAM memory when executing programs. A store instruction is said to be silent if it writes to memory location a value that is already present there. An important property of our approach is its portability to different processor architectures, contrarily to the previous hardware-level approach. We conducted a comprehensive evaluation of our proposal on the Rodinia benchmark. For that, we applied an analytic evaluation of the tradeoff between the silentness threshold of stores in a given program and the energy cost ratio of memory accesses. Depending on the silentness of evaluated applications and typical ratios, the gain in memory consumed by memory can reach up to 42 %. While this paper mainly targeted the STT-RAM technology (due to its maturity), the proposed silent store elimination applies as well to other NVMs with asymmetric access latencies.

This work will be extended by considering existing cycle-accurate simulation tools, combined with power estimation tools to evaluate more precisely the gain expected from memory configurations identified as the most energy-efficient with the present analytic approach. A candidate framework is MAGPIE [6], built on top of gem5 [3] and NVSim [7].

6. ACKNOWLEDGEMENTS

This work is funded by the French ANR agency under the grant ANR-15-CE25-0007-01, CONTINUUM project.

7. REFERENCES

[1] Gordon B Bell, Kevin M Lepak, and Mikko H Lipasti. Characterization of silent stores. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2000.

[2] Luca Benini and Giovanni de Micheli. System-level power optimization: Techniques and tools. *ACM Trans. Des. Autom. Electron. Syst.*, 5(2), April 2000.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[4] Shuai Che, Jeremy W. Sheaffer, Michael Boyer, Lukasz G. Szafaryn, Liang Wang, and Kevin Skadron. A characterization of the rodinia benchmark suite with comparison to contemporary CMP workloads. In *International Symposium on Workload Characterization (IISWC'10)*, 2010.

[5] Wei-Kai Cheng, Yen-Heng Ciou, and Po-Yuan Shen. Architecture and data migration methodology for L1 cache design with hybrid SRAM and volatile STT-RAM configuration. *Microprocessors and Microsystems*, 42, 2016.

[6] Thibaud Delobelle, Pierre-Yves Peneau, Abdoulaye

Gamatie, Florent Bruguier, Gilles Sassatelli Sophiane Senni, and Lionel Torres. Magpie: System-level evaluation of manycore systems with emerging memory technologies. In *Workshop on Emerging Memory Solutions - Technology, Manufacturing, Architectures, Design and Test at Design Automation and Test in Europe - DATE'2017, Lausanne, Switzerland*, 2017.

[7] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P. Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 31(7):994–1007, 2012.

[8] Ricardo Gonzalez and Mark Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of solid-state circuits*, 31(9):1277–1284, 1996.

[9] Jingtong Hu, Chun Jason Xue, Wei-Che Tseng, Yi He, Meikang Qiu, and Edwin H.-M. Sha. Reducing write activities on non-volatile memories in embedded CMPs via data migration and recomputation. In *Design Automation Conference (DAC'10)*, 2010.

[10] K Ikegami, H Noguchi, C Kamata, M Amano, K Abe, K Kushida, E Kitagawa, T Ochiai, N Shimomura, A Kawasumi, H Hara, J Ito, and S Fujita. A 4ns, 0.9v write voltage embedded perpendicular stt-mram fabricated by mtj-last process, 04 2014.

[11] Adwait Jog, Asit K. Mishra, Cong Xu, Yuan Xie, Vijaykrishnan Narayanan, Ravishankar Iyer, and Chita R. Das. Cache revive: architecting volatile STT-RAM caches for enhanced performance in CMPs. In *Annual Design Automation Conference DAC*, 2012.

[12] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, pages 75–, Washington, DC, USA, 2004. IEEE Computer Society.

[13] Kevin M Lepak, Gordon B Bell, and Mikko H Lipasti. Silent stores and store value locality. *Transactions on Computers*, 50(11), 2001.

[14] Kevin M. Lepak and Mikko H. Lipasti. On the value locality of store instructions. In *International Symposium on Computer Architecture (ISCA)*, 2000.

[15] Jianhua Li, Chun Jason Xue, and Yinlong Xu. STT-RAM based energy-efficiency hybrid cache for CMPs. In *International Conference on VLSI and System-on-Chip (VLSI-SoC'11)*, 2011.

[16] Qingan Li, Jianhua Li, Liang Shi, Chun Jason Xue, and Yanxiang He. MAC: Migration-aware compilation for STT-RAM based hybrid cache in embedded systems. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2012.

[17] Qingan Li, Jianhua Li, Liang Shi, Mengying Zhao, Chun Jason Xue, and Yanxiang He. Compiler-assisted STT-RAM-based hybrid cache for energy efficient embedded systems. *Transactions on Very Large Scale Integration (VLSI) Systems*, 22(8), 2014.

[18] Qingan Li, Liang Shi, Jianhua Li, Chun Jason Xue, and Yanxiang He. Code motion for migration minimization in STT-RAM based hybrid cache. In *Computer Society Annual Symposium on VLSI*, 2012.

[19] Qingan Li, Yingchao Zhao, Jingtong Hu, Chun Jason Xue, Edwin Sha, and Yanxiang He. MGC: Multiple graph-coloring for non-volatile memory based hybrid scratchpad memory. *Workshop on Interaction between Compilers and Computer Architectures*, 2012.

[20] Sparsh Mittal and Jeffrey S. Vetter. A survey of software techniques for using non-volatile memories for storage and main memory systems. *Trans. Parallel Distrib. Syst.*, 27(5), 2016.

[21] Hiroki Noguchi, Kazutaka Ikegami, Keiichi Kushida, Keiko Abe, Shogo Itai, Satoshi Takaya, Naoharu Shimomura, Junichi Ito, Atsushi Kawasumi, Hiroyuki Hara, and Shigeji Fujita. 7.5 a 3.3ns-access-time 71.2w/mhz 1mb embedded stt-mram using physically eliminated read-disturb scheme and normally-off memory architecture, 02 2015.

[22] Xiang Pan and Radu Teodorescu. Nvsleep: Using non-volatile memory to enable fast sleep/wakeup of idle cores. In *International Conference on Computer Design, ICCD*, 2014.

[23] Preeti Ranjan Panda, Nikil D. Dutt, Alexandru Nicolau, Francky Catthoor, Arnout Vandecappelle, Erik Brockmeyer, Chidamber Kulkarni, and Eddy de Greef. Data memory organization and optimizations in application-specific systems. *Design & Test of Computers*, 18(3), 2001.

[24] Clinton W. Smullen, Vidyabhushan Mohan, Anurag Nigam, Sudhanva Gurumurthi, and Mircea R. Stan. Relaxing non-volatility for fast and energy-efficient STT-RAM caches. In *International Symposium on High Performance Computer Architecture*, 2011.

[25] Guangyu Sun, Xiangyu Dong, Yuan Xie, Jian Li, and Yiran Chen. A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In *International Conference on High-Performance Computer Architecture (HPCA)*, 2009.

[26] Hung-Wei Tseng and Dean M. Tullsen. Data-triggered threads: Eliminating redundant computation. In *International Conference on High-Performance Computer Architecture (HPCA)*, 2011.

[27] Hung-Wei Tseng and Dean M. Tullsen. Software data-triggered threads. In *Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA*, 2012.

[28] Hung-Wei Tseng and Dean M. Tullsen. CDTT: compiler-generated data-triggered threads. In *International Symposium on High Performance Computer Architecture HPCA*, 2014.

[29] Shasha Wen, Milind Chabbi, and Xu Liu. REDSPY: exploring value locality in software. In *International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS*, 2017.

[30] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, and Yuan Xie. Power and performance of read-write aware hybrid caches with non-volatile memories. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2009.

[31] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. Energy reduction for STT-RAM using early write termination. In *International Conference on Computer-Aided Design*, ICCAD, 2009.

L

HEPSYCODE-RT : a Real-Time Extension for an ESL HW/SW Co-Design Methodology

HEPSYCODE-RT: a Real-Time Extension for an ESL HW/SW Co-Design Methodology

Vittoriano Muttillo
Center of Excellence DEWS
Via Vetoio, 1
L'Aquila, Italy
vittoriano.muttillo@graduate.univaq.it

Giacomo Valente
Center of Excellence DEWS
Via Vetoio, 1
L'Aquila, Italy
giacomo.valente@graduate.univaq.it

Daniele Ciambrone
Center of Excellence DEWS
Via Vetoio, 1
L'Aquila, Italy
daniele.ciambrone@student.univaq.it

Vincenzo Stoico
Center of Excellence DEWS
Via Vetoio, 1
L'Aquila, Italy
vincenzo.stoico@student.univaq.it

Luigi Pomante
Center of Excellence DEWS
Via Vetoio, 1
L'Aquila, Italy
luigi.pomante@univaq.it

ABSTRACT

This work [1] focuses on the definition of a methodology for handling embedded real-time applications, starting from an existing HW/SW co-design methodology able to support the design of dedicated heterogeneous parallel systems. The state-of-the-art related to similar tools and methodologies is presented and the reference framework with the proposed extension to the real-time world is introduced. A case study is then described to show a design space exploration able to consider such an extension.

CCS CONCEPTS

• **Hardware → Software tools for EDA**; • **Hardware → Modeling and parameter extraction**; *Timing Simulation*; *DSE;*

KEYWORDS

HW/SW Co-Design, Heterogeneous Parallel Systems, DSE, real-time systems

1 INTRODUCTION

During the last years, the spread and importance of embedded systems are increasing but it is still not yet possible to completely engineer their system-level design flow. Designers commonly adopt one or more system-level models (e.g. block diagrams, UML, *SystemC*, etc.) to have a complete problem view, to perform a check on HW/SW resources allocation and to validate the design by simulating the system behavior. In this scenario, SW tools to support designers to reduce costs and overall complexity of systems development are even more of fundamental importance. Unfortunately, there are no fully engineered general methodologies defined for this purpose and often the best option is still to refer to experienced designer indications to take advantage of empirical criteria and qualitative assessments. For example, systems based on heterogeneous multi-processor architectures (*Heterogeneous Multi-Processor Systems*, HMPS) have been recently exploited for a wide range of application domains, especially in the *System-on-Chip* (SoC) form factor (e.g. [2]). In particular, such architectures are often used to implement *Dedicated Systems* (DS), i.e. digital electronic systems with an application-specific HW/SW architecture designed to satisfy a priori known application with F/NF requirements. In such a case, they are called *Dedicated Heterogeneous Multi-Processor Systems* (D-HMPS). D-HMPS are so complex that the adopted *HW/SW Co-Design Methodology* plays a major role in determining the success of a product. The situation is even worse if the considered system is a hard/soft real-time one. In such a case, time constraints are normally defined considering the worst possible case (hard) or an average situation (soft).

In such a context, this work focuses on the definition of a HW/SW co-design methodology and the development of a related prototypal tool to improve the design time of embedded real-time applications. Specifically, the whole framework drives the designer from an *Electronic System-Level* (ESL) behavioral model, with related NF requirements, including real-time ones, to the final HW/SW implementation, considering specific HW technologies, scheduling policies and *Inter-Process Communication* (IPC) mechanisms. The remainder of the paper is organized as follows: *Section II* provides an overview of HW/SW co-design tools related to embedded real-time computing systems. *Section III* describes the reference HW/SW Co-Design methodology, whereas *Section IV* discusses the extension to adapt it to the real-time world. *Section V* presents a case study that shows a design space exploration able to consider such an

RAPIDO, January 22-24, 2018, Manchester, United Kingdom

ACM ISBN 978-1-4503-6417-1/18/01…$15.00
https://doi.org/10.1145/3180665.3180670

extension. Finally, *Section VI* reports some conclusive considerations and presents future works.

2 HW/SW CO-DESIGN OF REAL-TIME EMBEDDED SYSTEMS

A remarkable number of research works have focused on the system-level HW/SW co-design of D-HMPS [3]. In such a context, the most critical steps are always related to the *System Specification* and the *Design Space Exploration* (DSE) activities [4]. The main differences between the approaches are related to the amount of information and actions explicitly requested to the designer and influenced by his experience. In particular, many approaches (especially those based on the *Y-Chart* principle [5]) explicitly require as an input the HW architecture to be considered for mapping purposes. Other works [6] aims to the problem of designing embedded real-time systems starting from the input/output constraints on the final implementation. Offline schedulability and feasibility analysis involve different research works [7][8], with respect to the correct algorithms that can guarantee optimality depending on the load parameters. To analyze the behavior of a system, many tools have been developed to evaluate/estimate timing parameters, to validate scheduling and to perform simulations. In such a domain, the work presented in [9] starts from three sub-models, considering a model for SW application (*Platform Independent Model*) on one side and a platform (*Platform Description Model*) on the other side, and both models are connected by a *Platform Specific Model* that defines the mapping of SW into HW. By exploiting a specific extension for DSE and performance evaluation [10], in order to consider non-functional properties such as real-time, power, temperature, reliability constraints and so on, the tool offers different simulation and estimation outputs that drive the designer from the system-level model to the final implementation.

With respect to works that heavily relies on *Model of Computations* (*MoC*) theory, *ForSyDe* (*Formal System Design*) [11] is a methodology for modeling and design heterogeneous embedded and cyber-physical systems. The starting application is modeled by a network of processes interconnected by signals. Then, the model is refined by different design transformations into a target implementation language.

An interesting academic tool is *SynDEx* [12], a system level EDA tool based on the *Algorithm-Architecture Adequation* (*AAA*) methodology intended to find implementation solution, under real-time constraints, for embedded applications onto multi-component HW/SW architectures.

Finally, to have a look also to a SystemC-based commercial product, it is worth noting to cite *Intel CoFluent* [13] as a promising system-level modeling and simulation environment. Other than the model of the system behavior, it explicitly requires a manual modeling of both the hardware architecture and the mapping.

So, at the best of our knowledge, there are very few system-level HW/SW co-design methodologies that try to fully address the problem of both "automatically suggest an HW/SW partitioning of the system specification" and "map the partitioned entities onto an automatically defined heterogeneous multi-processor architecture" while considering also real-time constraints.

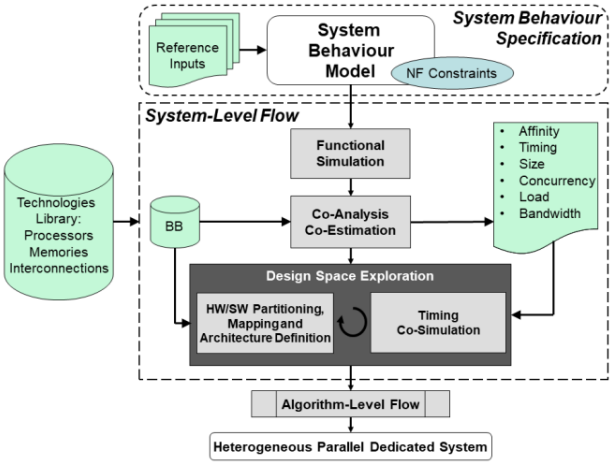


Figure 1: Reference HW/SW Co-Design Flow.

3 HW/SW CO-DESIGN FRAMEWORK

In the context of embedded real-time systems design, this work starts from a specific framework (called *HEPSYCODE*: *HW/SW Co-Design of Heterogeneous Parallel Dedicated Systems*) [14], based on an existing *System-Level HW/SW Co-Design Methodology* [15][19][21], and introduces the possibility to specify real-time requirements in the set of non-functional ones (the new framework is so called *HEPSYCODE-RT*). The main items composing such a methodology and its extension are discussed in the next paragraphs, while the reference ESL HW/SW Co-Design Flow is shown in Fig. 1.

3.1 Modeling Language

The system behavior modeling language introduced in *HEPSYCODE-RT*, named *HML* (*HEPSY Modeling Language*) [17], is based on the *Communicating Sequential Processes* (*CSP*) *MoC* [16]. It allows modeling the behavior of the system as a network of processes communicating through unidirectional synchronous channels. By means of HML it is possible to specify the *System Behavior Model* (*SBM*), an executable model of the system behavior, a set of *Non–Functional constraints* (*NFC*) and a set of *Reference Inputs* (*RI*) to be used for simulation-based activities. It is worth noting that another *HEPSYCODE* extension able to exploit more formal approaches is currently under development [17].

In particular *SBM = {PS, CH}* is a CSP-based executable model of the system behavior that explicitly defines communication among processes (*PS*) using unidirectional point-to-point blocking channels (*CH*) for data exchange. $PS = \{ps_1, ps_2, .. , ps_n\}$ is a set of concurrent processes that communicate

with each others exclusively by means of channels and use only local variables. Each process is described by means of a sequence of statements by using a suitable modeling language. Each process can have a priority *p*: 1 (lower) to 100 (higher) imposed by the designer. The concept of statement has to be fixed once selected a proper specification/modeling language. Languages suitable to describe CSP are *SystemC* (chosen for this work), *OCCAM*, *Handel-C*, *ADA* and so on. More abstract languages are *UML*, *SysML*, *Simulink* and so on. $CH = \{ch_1, ch_2, .. , ch_n\}$ is a set of channels where each channel is characterized by source and destination processes, and some details (i.e. *size*, *type*) about transferred data. Each channel can have also a priority *p*: 1 (lower) to 100 (higher) imposed by the designer.

$RI: \{(i_1, o_1), \ldots, (i_n, o_n)\}$ is a set of inputs (possibly timed), representative as much as possible of typical operating conditions of the system, and related expected outputs to be used for analysis and simulation-based validation.

The *Non–Functional Constraints* (*NFC*) are composed of *Timing Constraints (TC)*, *Architectural Constraints (AC)* and *Scheduling Directives*. Two different *TC* can be considered by the designer: *Time-to-Completion* (*TTC*), unique and related to the whole SBM, is the time available to complete the SBM execution from the first input trigger to the complete output generation; *Time-to-Reaction* (*TTR*) is a set of real-time constraints related to the time available for the execution of leaf CSP processes (i.e. the time available to execute the statements inside an input/output pair that delimits the CPS process main body, see Fig. 2). Different leaf processes can have different associated TTR. This real-time constraints are not strictly related to classical RT requirements, but impose a timing bound to the execution of some specific processes. Both TTC and TTR constraints shall be satisfied by each element of RI.

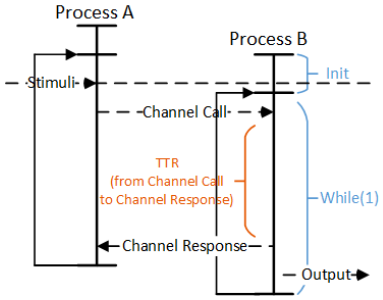


Figure 2: Time-To-Reaction Constrain.

The Architectural Constraints (AC) are related to the *Target Form Factor* (TFF) as *On-chip* (ASIC, FPGA) or *On-Board* (PCB) and to the *Target Template Architecture* (TTA) as type of available *Basic Blocks* (BB), min/max number of possible BB instances, min/max number of available *Interconnections* instances and/or total available area (or an equivalent metric). Finally, the *Scheduling Directives* (SD) specify the available scheduling policies. At the moment they are *First-Come First-Served* (FCFS) and *Fixed Priority* (FP) preemptive scheduling.

3.2 Technologies Library and Basic Blocks

The target HW architectures is composed of different basic HW components. These components are collected into a *Technologies Library* (*TL*). *TL* can be considered as a generic "database" that provides the characterization of the available technologies. In particular $TL = \{PU, MU, EIL\}$, where $PU = \{pu_1, pu_2, .. , pu_p\}$ is a set of *Processing Units*, $MU = \{mu_1, mu_2, .. , mu_m\}$ is a set of *Memory Units* and $EIL = \{il_1, il_2, .. , il_c\}$ is a set of *External Interconnection Links*. However, the detailed characterizations are dependent on *TFF*. The main differences are related to the different attributes (or different meaning of the same attribute) needed to characterize processing units, local memories, and interconnections. This work considers only *TL* for *PCB* where each PU that executes SW shall be a separate discrete *Commercial Off-The-Shelf* (*COTS*) *Integrated Circuit* (*IC*) mounted on a board.

PU elements are divided into two main groups: the ones that perform processing by means of the execution of some *Instruction Set Architecture* (ISA), called *SW-PU*, and the ones that perform processing without relying on an *ISA*, called *HW-PU*. Each pu_i in *PU* for *PCB* is characterized by a *Name*, a *Processor Type*, *Capacity* (SW-PU: max allowed *load*; HW-PU: available resources as number of equivalent-gates, *LUT*, Cell, etc.), *ISA* (only for SW-PU), *Frequency*, *Context Switch Overhead* (only for *SW-PU*), a *statement-level performance metric* (like CC4CS [18] or equivalent ones), and a *unit cost* (€). With respect to *Processor Type*, *PU* elements are further classified in three classes [1]: *General-Purpose Processors* (SW-PU: GPP); *Application-Specific Processors* (SW-PU: ASP) targeted to particular application domains (e.g. *Digital Signal Processors*, DSP); *Single-Purpose Processor* (HW-PU: SPP; realized by means of ASIC/FPGA.

MU elements are divided in two main classes: *Volatile Memory Units* (*VLMU*) and *Non-Volatile Memory Units* (*NVLMU*), with a main parameter related to capacity (i.e. bytes).

EIL elements are characterized by some parameters related to *bandwidth*, *number of connectable items* and *concurrency properties*.

The designer will use such components to build a set of *Basic Blocks* (BB). So, $BB = \{bb_1, bb_2, .. , bb_b\}$ is the set of BB available during DSE step to automatically define the HW architecture. A generic *BB* is composed of a set of *PU*, a set of *MU* and a *Communication Unit* (CU). *CU* represents the set of *EIL* that can be managed by a BB. BB internal architecture is dependent on *TFF* and *TTA*. In particular, each BB element can be generally composed of 1 or more *PU* elements, some *MU* elements and 1 *CU* element. *BB* elements are the ones effectively taken as input by the system-level flow for analysis, estimations and DSE steps. So, the target HW architecture can be seen as a set of *BB* elements interconnected by means of one or more *EIL* elements. The type of available BB are automatically defined by the selected *TTA*.

This work currently focuses only on: *Homogeneous Multi-Processor System with Distributed Memory* where each *BB* element is composed of only 1 *PU* element (homogenous among

BB elements), some local *MU* elements and 1 *CU* element; *Heterogeneous Multi-Processor System with Distributed Memory* where each *BB* element is composed of only 1 *PU* element (heterogeneous among BB elements), some local *MU* elements and 1 *CU* element.

3.3 ESL HW/SW Co-Design Flow

The first step of the adopted co-design flow is the *Functional Simulation* where *SBM* is simulated to check its correctness with respect to *RI*. Then, the next step aims at extracting as much as possible information about the system by analyzing the *SBM* while considering the available *BB*. This step is supported by *Co-Analysis* and *Co-Estimation* activities to evaluate/estimate several metrics related to the *BB* involved in the design flow.

Co-Analysis performs evaluation of two metrics. The first one is called *Affinity* [19]. The *Affinity* $A = \{[a_1, a_2, .. a_n] \mid a_i = [A(GPP_i), A(DSP_i), A(SPP_i)]\}$ of a process ps_i is a triplet of values in the interval [0,1] that provides a quantification of the matching between the structural and functional features of the functionality implemented by a process and the architectural features of each of *GPP*, *DSP*, and *SPP*. The second metric evaluated during *Co-analysis* is related to *Concurrency*. The *concurrency* $CN(PS, CH) = [CN_{PS}, CN_{CH}]$ is expressed by the set of processes *PS* and channels *CH* pairs that could be potentially concurrently "working", where $CN_{PS} = \{ [ps_i, ps_j] : ps_i \wedge ps_j$ *could be potentially executed concurrently}* and $CN_{CH} = \{ [ch_i, ch_j] : ch_i \wedge ch_j$ *could potentially transfer data concurrently}*. *CN* is evaluated by means of a functional simulation with respect to *RI*.

Co-estimation performs two kinds of estimations: *Static Estimations* of *Timing* and *Size*, *and Dynamic Estimations* of *Load* and *Bandwidth*. The *Timing* metric is the number of clock cycles needed to execute each statement *j* of each process ps_i by means of each processor *k* in the available BB, with *k=1..n*. The goal is to estimate how many clock cycles are needed by a specific BB to execute the implementation of a specific statement (e.g. [18] and [20] presents two possible approaches).

Size is a set of estimations for each statement of each process with respect to each available processor. It is related to bytes or area/resources metrics depending on SW or HW implementations. *L* is the *Load* (i.e. the *processor utilization percentage*) that each process would impose to each *not-SPP* processor to satisfy imposed TTC/TTR timing constraints (see *Section IV*). Finally the *Bandwidth* (B) is the number of bits sent/received over each channel (i.e. bits exchanged by communicating processes pairs in PS) during an interval of time equal to TTC.

After this steps, the reference co-design flow reaches the DSE step (as shown in Figure 3). It includes two iterative activities: "*HW/SW Partitioning, Mapping and Architecture Definition*", based on a genetic algorithm that allows to explore the design space looking for feasible mapping/architecture items suitable to satisfy imposed constraints; "*Timing Co-Simulation*", that considers suggested mapping/architecture items to actually check for timing constraints satisfaction. When the mapping/architecture item proposed by the DSE step is acceptable, it is possible to proceed with system implementation (i.e. *Algorithm-Level Flow*).

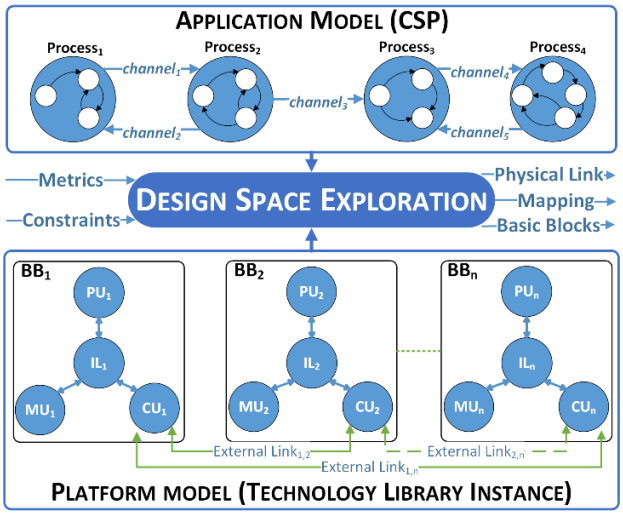


Figure 3: Design Space Exploration Framework.

4 HEPSYCODE-RT: PROPOSED EXTENSION

With respect to NF requirements, this work provides an extension that allow the methodology to better consider architectural and timing constraints. Related to the *SBM* model, it is now possible to identify two classes of CSP processes: *classical CSP process* and *real-time CSP processes*. In the current version, the last ones shall be leaf processes and their body (i.e. a never-ending loop) shall start with a channel read and end with a channel write towards the same process. To such input/output pair will be referred the *TTR* constraint. Moreover, in such a context, a *CSP to Task Model* transformation has been defined to allow considering classical real-time world notations. Such a transformation involves concepts related to both processes and channels.

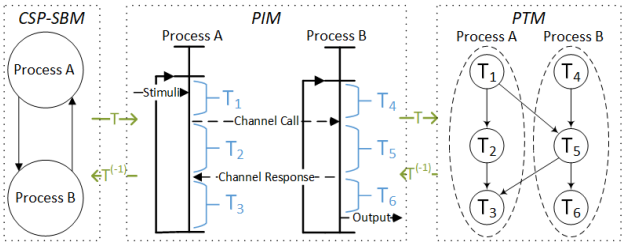


Figure 4: Time-To-Reaction Constrain.

The general transformation is shown in Figure 4. In this example the *CSP SBM* model is first expanded in a *Process Interaction Model* (*PIM*), where the processes A and B are split into different pieces of code, delimited by channel calls. The final transformation starts from the *PIM* model and associates the single pieces of code to specific tasks in the classical task representation models (i.e. *Process-Task Model*, *PTM*). At this time, the designer should write a SBM avoiding cycles to match the classical real-time DAG representation of tasks.

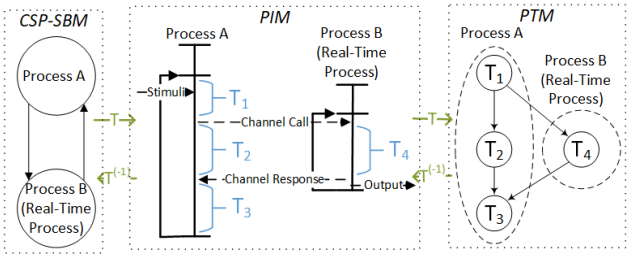


Figure 5: Time-To-Reaction Constrain.

With respect to the real-time CSP processes, the actual transformation is the one shown in Figure 5. With this specific kind of representation it is possible to consider concurrently timing constraints related to the whole SBM (TTC) and real-time constraints related to the reaction of specific processes (TTR) while considering periodic leaf processes as periodic ones.

With this assumption, it is possible to adapt the *Load Estimation* step to consider such real-time constraints. In particular, the load can be defined in two different ways.

The *Load* L_i that each **non real-time process** ps_i would impose to each *non-SPP* processor *s* to satisfy TTC. L_i is estimated by allocating all the *n* processes to a single-instance of each software processor *s* ($pu_j \subseteq \{[pu_1, .. , pu_s]\}$ *with* $s \leq$ *Total Number of* pu_j) and performing some simulations. Three parameters have to be computed: FRT_j *(Free Running Time), i.e.* the total application simulation time on processor pu_j; t_i, the simulated time for each process ps_i in a loop on processor pu_j; *N*, the number of simulation loops. Starting from this estimated parameters, the *Free Running Load* FRL_i is calculated by the equation:

$$FRL_i = \frac{(t_i * N)}{FRT_j} \quad (1)$$

where FRT_j/N is the average period of each processes on processor pu_j. By imposing that the execution time shall be equal to *TTC*, it is possible to evaluate the Load L_i that processes ps_i would impose to the *SW* processor to satisfy *TTC* itself. In fact, setting FRT_j equal to *TTC*, for each process/processor pair, such as:

$$\mathrm{TTC} = x_j * FRT_j \qquad with\ 0 \leq x_j \leq 1 \quad (2)$$

The value of estimated Load L_i that the system imposes to processor pu_j to satisfy *TTC* is:

$$\mathrm{L}_i = \frac{(t_i * N)}{TTC} = \frac{(t_i * N)}{FRT_j} * \frac{FRT_j}{TTC} = \frac{FRL_j}{x_j} \quad (3)$$

The Load L_i that each **real-time process** ps_i would impose to each *s* software processor to satisfy input real-time constrain TTR_i is directly set equal to:

$$L_i = \frac{t_i}{TTR_i} \quad (4)$$

TTR_i is the real-time constraint related to the process ps_i. In this way it is possible to consider two different situations: *Hard real-time process*, if $t_i < TTR_i$, the constraints are fulfilled and it is possible to consider the value L_i as an input to the DSE step; *Soft real-time process*, if $t_i < (TTR_i + \delta(t))$, then constraints could be considered as soft real-time ones.

Then, thanks to all the estimated TTC/TTR loads, it is possible to perform DSE step in order to fulfill also RT constraints. Moreover, an additional architectural constraint deriving from TTR is that non-SPP processors executing real-time processes have to adopt a scheduling policy suitable for real-time scheduling (e.g. *fixed-priority preemptive scheduling*). Finally, the effect of such scheduling policy shall be considered during the timing co-simulations performed to validate the proposed solutions.

5 CASE STUDY

This section presents a simple case study used to show the effects of the proposed real-time extension to HEPSYCODE.

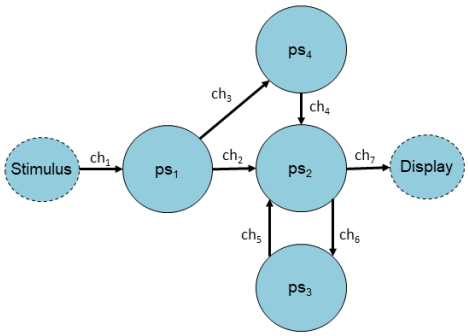


Figure 6: CSP MoC Example

The reference SBM is shown in Figure 6, where the processes $PS = \{ps_1, .. , ps_4\}$, with priority of $\{ps_1, ps_2, ps_4\}$ equal to each other and priority of ps_3 higher than the others, exchange data using the channels $CH = \{ch_1, .. , ch_7\}$. In this scenario there are three non real-time processes $\{ps_1, ps_2, ps_4\}$ and one process $\{ps_3\}$ with real-time constraint equal to TTR_3. The whole SBM is also subject to a TTC. So, for a given processor pu_j, the load parameters for the four processes are: $L_{1,j}=t_{1,j}/([x_j*FTR_j]/N)$, $L_{2,j}=t_{2,j}/([x_j*FTR_j]/N)$, $L_{4,j}= t_{4,j}/([x_j*FTR_j] /N)$, $L_{3,j} = t_{3,j}/TTR_3$. After the *Co-analysis* and *Co-estimation* steps, by considering such loads, the DSE step should be able to suggest a partition/mapping item able to fulfill both TTC and TTR_3 constraints. Several DSE have been performed considering different TTC/TTR pairs. In the considered use case the available BBs are: bb_1 (SW-PU): 20 MHz 8-bit 8051 CISC core with 128 byte of Internal RAM, 64K of internal ROM, without cache and external memory (cost 10); bb_2 (SW-PU): 150 MHz 32-bit LEON3 soft-processor with 2*4 KiB L1 caches, RAM size of 4096 KiB and a ROM of 2048 KiB (cost 50); bb_3 (HW-PU): 300 MHz Altera Stratix V (cost 300).

For each BB is allowed maximum 1 instance and they are supposed to communicate by means of a shared bus. Moreover, each SW-PU uses a *Fixed Priority* preemptive scheduling algorithm. Results shown in Table 1 figure out as the *DSE* step with real-time extension is able to satisfy *TTC/TTR* constraints, at

least with respect to timing simulations. In particular, by setting *TTR* and decreasing *TTC*, *DSE* suggests solutions that fulfil the timing requirements most of the time (two not satisfactory suggestions are underlined in Table 1). Decreasing the *TTR*, the *DSE* suggests to allocate the real-time process on pu_j that fulfil the constraints. It is worth noting that, if the *TTR* is very strict, the only valid mapping involve the use of a more expensive FPGA.

Table 1: Results from the DSE on the Use Case Example

Allocation	Simulated Time (ms)	ps_3 (ms)	TTC (ms)	TTR (ms)
All on bb_1	794,88	8,10	*600*	10
All on bb_2	650,66	5,54	*600*	10
ps_1, ps_2, ps_3 on bb_1 ps_4 on bb_2,	590,80	8,10	*600*	10
ps_3 on bb_1 ps_1, ps_4 on bb_2 ps_2 on bb_3	264,89	8,10	*400*	10
ps_1, ps_3 on bb_1 ps_4 on bb_2 ps_2 on bb_3	298,88	8,10	*300*	10
ps_4 on bb_1 ps_1, ps_2, ps_3 on bb_3	201,48	0,009	*200*	10
ps_3 on bb_1 ps_4 on bb_2 ps_1, ps_2 on bb_3	145,67	8,10	*200*	10
ps_3 on bb_1 ps_1, ps_2, ps_4 on bb_3	81,04	8,10	*100*	10
ps_1, ps_2 on bb_1 ps_3, ps_4 on bb_2,	562,85	5,54	*600*	7
ps_1, pu_4 on bb_1 ps_2, ps_3 on bb_2	462,58	5,54	*400*	7
ps_1 on bb_1 ps_3, ps_4 on bb_2 ps_2 on bb_3	220,80	5,54	*400*	7
ps_1, on bb_1 ps_3 on bb_2 ps_2, ps_4 on bb_3	206,99	5,54	*300*	7
ps_3 on bb_2 ps_1, ps_2, ps_4 on bb_3	55,55	5,55	*200*	7
ps_1, ps_4 on bb_1 ps_2 on bb_2 ps_3 on bb_3	428,87	0,009	*600*	4
ps_4 on bb_1 ps_1, ps_2 on bb_2 ps_3 on bb_3	337,56	0,009	*400*	4
ps_4 on bb_1 ps_1 on bb_2 ps_2, ps_3 on bb_3	214,80	0,009	*300*	4
ps_4 on bb_2 ps_1, ps_2, ps_3 on bb_3	137,62	0,009	*200*	4
All on bb_3	0,22	0,009	*100*	4

4 CONCLUSIONS

This work has proposed an extended Electronic Design Automation (EDA) methodology (and related tools) in the ESL domain supporting the development of Real-time Embedded Systems. The final result is a methodology able to support real-time systems developments by suggesting both the platform and mapping solutions for the specific application. Future works will involve the introduction of other parameters associated to PU such as Power (peak power [W] or other metrics) and Energy. Others analysis, use cases and tests will be done in future, but starting from this preliminary results it is easy to note that the DSE step with load estimation and real-time extension seem to be quite effective with respect to execution times estimated by simulation. Validation on the final HW/SW implementation must be done in future to reduce errors at design time.

ACKNOWLEDGMENTS

This work has been partially supported by the ECSEL RIA 2016 MegaM@Rt2 and AQUAS projects.

REFERENCES

[1] Vahid, F. and Givargis, T. Embedded System Design: A Unified Hardware/Software Introduction. John Wiley & Sons, NY, USA, 2001.
[2] Xilinx Zynq7000, http://www.xilinx.com.
[3] Jia, Z. J., Bautista, T., Núñez, A. Pimentel, A. D. and Thompson, M. A system-level infrastructure for multidimensional MP-SoC design space co-exploration. In *ACM Trans. Embedd. Comput.* Syst. 13, 1s, Article 27 (December 2013), 26 pages, 2013.
[4] Teich, J. Hardware/Software Codesign: The Past, the Present, and Predicting the Future. Proceedings of the IEEE, vol. 100, no. Special Centennial Issue, pp. 1411-1430, 2012.
[5] Keutzer, K., Malik, S., Newton, A., Rabaey, J., and Sangiovanni-Vincentelli, A. System level design: Orthogonalization of concerns and platform-based design. IEEE Trans. Comput.-Aided Des. Integ. Circ. Syst. 19, 12, 1523–1543, 2000.
[6] Lee, E. A. and Seshia, S. A. Introduction to Embedded Systems, a Cyber-Physical Systems approach. In MIT Press, Second Edition, 2015.
[7] Real, J. and Crespo, A. Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. In Journal Real-Time Systems, 26, 2, 161-19, 2004.
[8] Buttazzo, G. Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications. In Springer, 3rd edition, 2011.
[9] Posadas, H., Penil, P., Nicolas, A. and Villar., E. Automatic synthesis of communication and concurrency for exploring component-based system implementations considering uml channel semantics. In Journal of Systems Architecture, 61, 8, 341-360, 2015.
[10] Contrex - Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties. https://contrex.offis.de.
[11] Rosvall, K. and Sander. I. A constraint-based design space exploration framework for real-time applications on MPSoCs. In Design, Automation and Test in Europe, Dresden, Germany. 2014.
[12] Yu, H., Ma, Y., Gautier, T., Besnard, L., Talpin, J.P., Le Guernic, P. and Sorel., Y. Exploring system architectures in aadl via polychrony and syndex. In Frontiers of Computer Science Journal, 7, 5, 627-649, 2013.
[13] Intel cofluent. http://www.intel.com.
[14] Hepsycode: A System-Level Methodology for HW/SW Co-Design of Heterogeneous Parallel Dedicated Systems, www.hepsycode.com.
[15] Pomante, L. System-level design space exploration for dedicated heterogeneous multi-processor systems. In Conf. on Appl. Syst., 79-86, 2011.
[16] Hoare, C. A. R. Communicating sequential processes. In Springer, New York, NY, 413-443, 1978.
[17] Di Pompeo, D., Incerto, E., Muttillo, V., Pomante, L. and Valente, G. An Efficient Performance-Driven Approach for HW/SW Co-Design. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering (ICPE '17), ACM, New York, NY, USA, 323-326, 2017.
[18] Stoico, V., Muttillo, V., Valente, G., Pomante, L. and D'Antonio, F. CC4CS: A Unifying Statement-Level Performance Metric for HW/SW Technologies, In Eur. Conf. on Digit. Syst. (DSD), 2017.
[19] Pomante, L., Sciuto, D., Salice, F., Fornaciari, W. and Brandolese, C. Affinity-Driven System Design Exploration for Heterogeneous Multiprocessor SoC, In IEEE Trans. on Comp., 55, 5, 2006.
[20] Allara, A., Brandolese, C., Fornaciari, W., Salice, F. and Sciuto, D. System-level performance estimation strategy for sw and hw, In Proc. Int. Conf. on Comp., Austin, TX, 48-53, 1998.
[21] Pomante, L. HW/SW Co-Design of Dedicated Heterogeneous Parallel Systems: an Extended Design Space Exploration Approach. In IET Computers & Digital Technique, 2013.

LVIII

NVMain Extension for Multi-Level Cache Systems

NVMain Extension for Multi-Level Cache Systems

Asif Ali Khan, Fazal Hameed and Jeronimo Castrillon
Chair For Compiler Construction
Technische Universität Dresden, Germany
{ asif_ali.khan, fazal.hameed, jeronimo.castrillon }@tu-dresden.de

ABSTRACT

In this paper, we present an extension of the NVMain memory simulator. The objective is to facilitate computer architects to model complex memory designs for future computing systems in an accurate simulation framework. The simulator supports commodity memory models for DRAM as well as emerging non-volatile memories technologies such STT-RAM, ReRAM, PCRAM and hybrid models. The current publicly available version of NVMain, NVMain 2.0, offers support for main memory (using DRAM and NVM technologies) and a die-stacked DRAM cache. We extend the cache model of the simulator by introducing an SRAM cache model and its supporting modules. With this addition, designers can model hybrid multi-level cache hierarchies by using the die-stacked DRAM cache and SRAM caches. We provide a reference implementation of an optimized cache organization scheme for die-stacked DRAM cache along with a tag-cache unit that, together, reduces cache miss latency. To enable integration of the new features in the existing memory hierarchy, we make necessary changes to the memory controller. We provide functional verification of the new modules and put forward our approach for timing and power verification. We run random mixes of the SPEC2006 benchmarks and observe $\pm 10\%$ difference in simulation results.

CCS Concepts

•**Memory System Design** → Memory System Hirarichy; Memory Technologies; •**Memory Simulators** → Simulator Accuracy; Design options; Simulator Speed; Extendibility; Resilience;

Keywords

Memory Simulator; SRAM Cache; Cache Organization; Row Buffer; Tag-cache

RAPIDO, January 22–24, 2018, Manchester, United Kingdom

978-1-4503-6417-1/18/01…$15.00

DOI: https://doi.org/10.1145/3180665.3180672

1. INTRODUCTION

CPU and memory system are the two most important components in any computing system. From the performance perspective, both components are strictly interlinked. CPU frequency has rapidly increased in the past decade while frequency of the memory system has not scaled up at the same pace. The memory system has always been on the down side due its high access time and relatively low operating frequency, becoming a major bottleneck in modern days computing systems. The rise of multi-core systems has further worsened this problem and the available per core capacity and per core bandwidth has diminished further. Commodity memory technologies such as DRAM are unable to fill this ever-increasing processor memory gap.

DRAM is highly criticized for being expensive in terms of power as well. The major factor that dominates DRAM power dissipation is its periodic refreshes, background and leakage power. The newly emerged non-volatile memory (NVM) technologies such as spin-torque-transfer random-access memory (STT-RAM), phase-change random-access memory (PCRAM), and resistive random-access memory (ReRAM) are believed to alleviate these limitations. They have been advocated as potential DRAM replacements at various levels (main memory, die-stacked cache). While these technologies could supplement or supersede conventional memory technologies at various levels in the memory hierarchy, they have their own limitations. Simply replacement of DRAM with NVM technologies is not a viable option because the latter suffers from high write energy and write endurance issues.

This opens up new research directions and calls for exploration of appropriate memory technologies at each memory level. An ideal memory system would use the best of many memory technologies and fulfill the diverse demands of modern days applications. To design such system, it is vital for computer architects to use simulation tools and study the suitability of each technology. NVMain [24] is one such simulator that provides support for both DRAM and NVM technologies. It models energy and cycle accurate operations of main memory system. In its extended version NVMain 2.0 [25], die-stacked DRAM cache was introduced with the Alloy [26] model.

In this paper, we present extensions for the publicly available version of NVMain (NVMain 2.0) simulator. We provide a reference implementation of a new SRAM cache which could be used at various cache levels. For die-stacked cache, we provide implementation of various latency optimizations. To demonstrate this extension, we model a recently proposed

high associativity cache organization called LAMOST [9]. A supporting module called *tag-cache* (an SRAM based small memory unit that stores the tag information of recently accessed sets as explained in [14, 9, 20]) is employed on top of LAMOST to mitigate high DRAM cache tag lookup latency. We modify the memory controller and other NVMain modules to make provision for the new extensions.

While some of these architectural features such as tag-cache support specific cache organizations [17, 9], others such as SRAM cache model can be used to model any level in the memory hierarchy. All the new modules are made configurable and can be enabled or disabled easily. Configuration parameters of each individual module are passed in a config file and can be varied as per design requirements. We believe these new features make NVMain more powerful and increase its application scope. Considering the design objectives, designers have more freedom to (a) model customized memory systems (b) choose memory technology for each level from a wider list of available options (DRAM, NVM, SRAM) (c) select the number of cache/memory levels (d) choose appropriate cache organization (Alloy, LAMOST, LAMOST with tag-cache) for die-stacked memory.

2. RELATED WORK

Simulation has become a powerful tool in computer architecture community that empowers designers to model their desired systems and predict the design objectives beforehand. Architectural simulation is mainly used for design space exploration and performance evaluation considering all design goals and performance parameters. As such, every leading processor manufacturing industry has developed its own simulation tool(s) to model new processor models and assess their feasibility. For instance, Mambo [4] by IBM is designed to model their past and future power designs. Its architectural support ranges from cell to embedded system to supercomputer (BlueGene, POWER7). Similarly, SimNow [2] by AMD and HAsim [23] by Intel are used to evaluate their future systems respectively. Some high speed architectural simulators have been developed to reduce the simulation time. Sniper [5], Graphite [21], SlackSimslacksim, P-Mambo [30] and COTSon [1] are to name a few. Open source system simulators such as Gem5 [3] exist that not only target micro-architecture of the processor but encompasses the whole system architecture. In a recent work [19], Gem5 has been coupled with SystemC that opens up a whole new set of options for system level design space exploration.

Unfortunately, some of these micro-architecture and system-level simulators do not model the memory system in detail. They consider a simplistic memory model without considering the complex organization of modern memory systems. More often, a fixed latency is associated with a memory access. In more advanced CPU simulators, bank conflicts are considered and every time it occurs, a fixed wait latency is added to the overall access time. This model significantly underestimates the actual characteristic of the memory system. This necessitates design of specialized memory simulators that consider all possible design features, model actual bus latencies, and report correct timing and energy parameters.

Of late, many such memory simulators have been published. DRAMSim [29], the first publicly available memory simulator, was designed to offer support for multiple types of

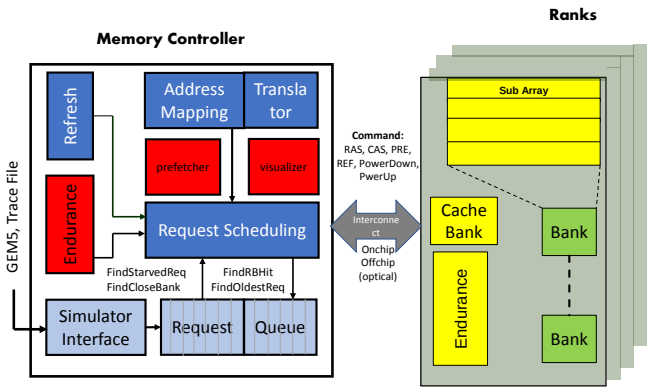


Figure 1: Overview of the NVMain Architecture [24]

memory technologies (SDRAM, DDR) and allow exploration of memory system concepts. Presently, its extended version DRAMSim2 [27] is openly available and is broadly used. It is a C++ based cycle-accurate model that provides most of the memory system design features. Design goals of the simulator were to keep it small and portable. However, this simplicity in controller architecture has barred it to be comparable with more recent and performance optimized controllers. Other memory simulators such as Ramulator [16], DRAMSys [15], NVMain and integrated simulators such as [28] have been proposed with different design goals. For instance, Ramulator offers support for an extended list of DRAM standards. DRAMSys provides a holistic framework that takes into consideration new DRAM based memory solutions such as JDEC DDR4, WIDE I/O, and HMC. It captures new aspects such as temperature and retention failures. In contrast to all these simulators, NVMain focuses on emerging NVM technologies while keeping the DRAM support intact. A brief comparison of these memory simulators with reference to design options is presented in Table 1.

3. SIMULATOR ARCHITECTURE

NVMain is a flexible memory simulator that supports DRAM and NVM technologies. For DRAM, it follows the same approach as other DRAM specific simulators in order to capture important features. For NVM devices, beside modeling timing and power parameters, it models NVM specific features as well; such as endurance, high write energy, and multi-level cells (MLC) capability. Major modules of the NVMain simulator include timing, power and endurance models. NVMain 2.0, the currently available version, extends support for sub-array level parallelism and MLC operations. *Object hooks* are introduced to strengthen the simulator hierarchy and allow requests inspection at a particular level.

3.1 NVMain Overview

NVMain is an object based model where every module is created as a separate object that can easily be integrated to or detached from the simulator. An overview of the base architecture of NVMain is shown in Figure 1.

Every object in the NVMain simulator such as the memory controller, the interconnect, the rank or the bank is responsible to model and capture its timing parameters. The

Table 1: Comparison of the Memory Simulators

Simulator	DRAM	die-stacked cache	NVM	Alloy [26]	LAMOST [9]	tag-cache	SRAM
DRAMSim2	✓	✗	✗	✗	✗	✗	✓
NVMain2.0	✓	✓	✓	✓	✗	✗	✗
Ramulator	✓	✗	✗	✗	✗	✗	✓
DRAMSys	✓	✗	✗	✗	✗	✗	✓
NVMainExt (proposed)	✓	✓	✓	✓	✓	✓	✓

timing parameters for DRAM devices are taken from their manufacturer data-sheets while for SRAM and NVM technologies, these parameters are obtained from CACTI [22] and NVSIM [8] tools respectively. The corresponding memory parameters such as t_{RCD}, t_{RP}, t_{BURST} are provided in a memory configuration file to the simulator. The simulator takes another configuration file that describes the overall memory system hierarchy and general configuration parameters such for number of ranks, banks, rows, columns, address mapping scheme, decoders, row buffer policies, queue models and queue sizes.

NVMain offers both single bank and inter-bank timing models. The single bank timing model tracks most commonly found parameters such as t_{RCD}, t_{RP} and t_{CAS} etc. The inter-bank timing model restricts the power consumption and current drawn by a single bank or different banks within a fix time period by introducing parameters like t_{FAW} (four activation window) and t_{RRD} (row to row activation delay). All timing parameters are considered before issuing a memory command to a particular module. Error message is generated in case a module violates the timings constraints.

In its power model, NVMain computes per device energy consumption where each device consists of multiple banks. The total energy consumption is calculated as the sum result of energy consumptions of all devices. For DRAM devices, the typical IDDx parameters are used to measure power consumption of read and write operations. For NVM technologies, to calculate the actual power consumption, NVMain takes per bit write energy from NVSIM. The endurance model keeps track of the total number of bits written and their values. Energy of the unchanged bits is subtracted from the total write energy consumption.

While the simulator provides a solid base for exploring the optimal memory system, the design features it offers are limited. The missing features include: a wide array of memory technologies, configurable number of memory levels, flexible cache controllers optimized for various performance parameters, tag-cache, predictors and multiple row buffer mapping schemes. The object-oriented structure of NVMain encourages development of new extensions and allow their easy integration.

3.2 NVMain Extensions

We provide reference implementations of new architectural features for NVMain that could be used to propose future memory systems. The new extensions strengthen the relatively simple cache model of the current NVMain simulator and opens up new design directions. Figure 2 highlights these features and demonstrates how they fit in the overall memory system design.

As highlighted in Figure 2, the die-stacked last level cache (LLC) can be modeled as DRAM or NVM technologies. For cache organization, designers have the choice to select appropriate scheme depending on the application requirements. Alloy cache [26] gives the best performance for applications having reduced miss-rate. Conversely, LAMOST [9] and LH [17] cache organizations perfom better than Alloy cache for applications having higher miss rates.

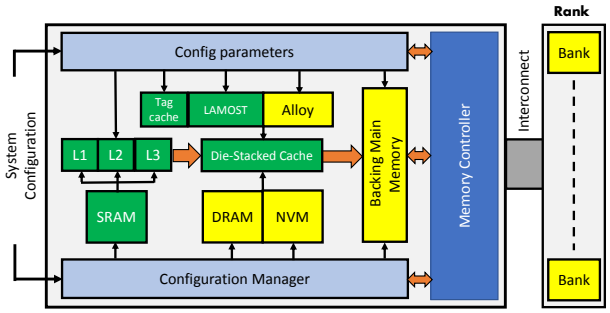


Figure 2: Extended NVMain Architecture (The green color highlights the extensions)

The proposed SRAM module can be used to model any cache level in the memory hierarchy. Typically, CPU simulators model only lower cache levels (level 1, 2 and 3). Existing NVMain simulator can be used to model die-stacked DRAM cache, typically used as LLC. Our extended version provides support to model multi-level cache hierarchy where lower cache levels can be realized with SRAM technology while higher cache level can be implemented using DRAM or NVM technology.

3.2.1 SRAM Cache Model

SRAM is a fast memory technology. It has small access time compared to DRAM and NVM technologies but is more expensive. It is used at lower cache levels (close to the processor) to bridge the processor and (slow)memory speed gap. NVMain simulator in its present form does not support SRAM. As a result, it can not model the lower cache levels. We introduce a reference model of SRAM cache which can be used as Level 1(L1), Level 2(L2) or Level 3(L3) cache.

To model the actual complex memory system, designers should define all cache levels in the memory hierarchy. Unfortunately, existing memory simulators offer only higher cache level(s). With this SRAM model, NVMain is capable to model the entire memory system by its own. Every parameter of the SRAM model is configurable and can be changed as per design goals. We adopt existing cache bank model of the simulator with varied parameters to implement SRAM specific functionalities. For every new request, the address translator of the SRAM module retranslates the physical address of the request and sets the SRAM related memory partitions. The SRAM cache is checked for

a hit/miss and based on the type of request, it is serviced accordingly (detailed discussion on the flow of commands to serve a read or write request is beyond the scope of this paper). A sub-request (e.g. cache line eviction, cache fill), if any, generated by this module is solely owned by it and has to be deleted after its completion. Further, this module forwards request to the next level in the memory hierarchy based on the system configuration. The next level, if exists, is added as a child to the SRAM module and can be accessed via an *object hook*. Appropriate requests and responses are generated between parent and child modules where parent corresponds to the current level and child corresponds to the next level in the memory hierarchy.

3.2.2 LAMOST Cache Organization

Presently, NVMain implements the Alloy cache organization for its die-stacked cache and it is by-default selected. Designers have no choice to change the cache organization. Alloy cache is a direct-mapped model that suffers from high miss rate and high off-chip memory latency. We implement a new cache organization that overcomes the limitations of the Alloy cache. For demonstration, we model LAMOST with a supporting tag-cache that reduces the cache hit/miss latency. In addition, we develop new row and set mapping schemes and make necessary changes to the memory controller. More details of the mapping schemes can be found in [10].

3.2.3 Tag Cache

Tag cache, fundamentally, is not a level in the memory system. Rather, it is a supporting unit for the die-stacked cache. Tag-cache stores the tag information of the most recently accessed cache sets. Future accesses to the same set(s) in die-stacked cache result in reduced access latency. Cache organizations such as ATCache [14] and LAMOST [9], when used with tag-cache, positively benefit from it. Detailed architectural benefits of tag-cache and various cache organizations can be found in [26, 14, 17, 10, 20]. For correctness, the simulator makes sure the right system configuration and generates an error otherwise.

4. EVALUATION

NVMain can be used in both trace based simulation as well full system simulation mode. For full system simulation, we connect NVMain to gem5 by applying the publicly available patch and run SPEC benchmarks on it. Gem5 uses the memory system modeled by NVMain instead of using its default memory model. An abstract overview of the full system configuration is presented in Figure 3. We use Simpoint [12] in order to reduce the benchmarks execution time. The idea is, to run a sub-set of instructions (for each benchmark) that imitates the behavior of the whole benchmark. In trace based simulation, we generate traces from Gem5 and provide trace file(s) as input to the NVMain simulator.

For functional verification of the new extensions, we use random mixes of the SPEC2006 [13] benchmarks. For the same system configuration, we run SPEC benchmarks in sim-zesto simulator [18] and NVMain. For ten experimental runs, we observe performance parameter (miss-rate) of L1, L2 and L3 caches and report the average measures in Table 2.

Table 2: Functional Verification of the New Extensions

Cache Level	sim-zesto normalized miss-rate	NVMain normalized miss-rate	Difference(%)
L1 (32 KB)	1.00	0.96	+4
L2 (256 KB)	1.00	1.09	−9
L3 (8 MB)	1.00	0.95	+5

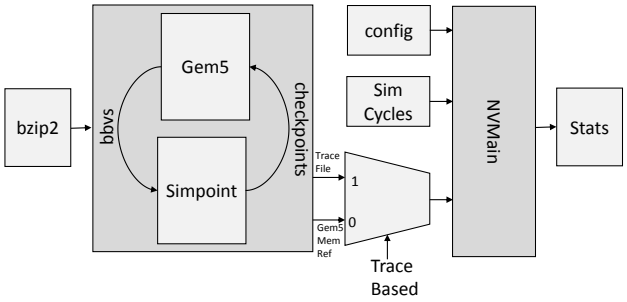


Figure 3: Full system simulation overview

Simulation results of the NVMain are in ±10% of the sim-zesto results. The memory model in sim-zesto is simplistic and can not accurately model LLC. Therefore, for the cache organization scheme LAMOST, we compare our results with the actual results presented in [9] and observe that they are in the accuracy range of ±5%.

For speed comparison, we run both sim-zesto and NVMain for 3B (3×10^9) instructions and observe that NVMain is around 10 times faster than sim-zesto. The timing and energy models of NVMain have already been verified. However, in future, we plan to verify these models for new extensions as well using Verilog model (for timing) and the publicly available DRAMPower2 [7] for (energy).

5. CONCLUSIONS

We have presented an extended version of the NVMain simulator. Considering the fundamental design goals — flexibility, simple user interface and scalability — of the simulator, we have provided reference implementations of SRAM cache, an optimized cache organization for die-stacked cache and a tag-cache model. The newly added design features widen the list of design options and enable customized design modelling. We have outlined the existing simulator architecture in brief and the new extensions in detail. We have run random mixes of the SPEC benchmarks in NVMain and observed that simulation results of the new extensions are in conformance with state-of-the-art. We will use this simulator to investigate memory systems with emerging memory technologies like [11] in the context of the Orchestration project [6].

Acknoledgments

This work was partially funded by the German Research Council (DFG) through the Cluster of Excellence 'Center for Advancing Electronics Dresden' (cfaed).

6. REFERENCES

[1] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega. Cotson: Infrastructure for full system simulation. *SIGOPS Oper. Syst. Rev.*, 43(1):52–61, Jan. 2009.

[2] R. Bedicheck. Simnow: Fast platform simulation purely in software. In *Hot Chips 16*, 2004.

[3] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[4] P. Bohrer, J. Peterson, M. Elnozahy, R. Rajamony, A. Gheith, R. Rockhold, C. Lefurgy, H. Shafi, T. Nakra, R. Simpson, E. Speight, K. Sudeep, E. Van Hensbergen, and L. Zhang. Mambo: A full system simulator for the powerpc architecture. *SIGMETRICS Perform. Eval. Rev.*, 31(4):8–12, Mar. 2004.

[5] T. E. Carlson, W. Heirman, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 52:1–52:12, New York, NY, USA, 2011. ACM.

[6] J. Castrillon, M. Lieber, S. Klüppelholz, M. Völp, N. Asmussen, U. Assmann, F. Baader, C. Baier, G. Fettweis, J. Fröhlich, A. Goens, S. Haas, D. Habich, H. Härtig, M. Hasler, I. Huismann, T. Karnagel, S. Karol, A. Kumar, W. Lehner, L. Leuschner, S. Ling, S. Märcker, C. Menard, J. Mey, W. Nagel, B. Nöthen, R. Peñaloza, M. Raitza, J. Stiller, A. Ungethüm, A. Voigt, and S. Wunderlich. A hardware/software stack for heterogeneous systems. *IEEE Transactions on Multi-Scale Computing Systems*, Nov. 2017.

[7] K. Chandrasekar, B. Akesson, and K. Goossens. Improved power modeling of ddr sdrams. In *Proceedings of the 2011 14th Euromicro Conference on Digital System Design*, DSD '11, pages 99–108, Washington, DC, USA, 2011. IEEE Computer Society.

[8] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(7):994–1007, July 2012.

[9] F. Hameed, L. Bauer, and J. Henkel. Simultaneously optimizing dram cache hit latency and miss rate via novel set mapping policies. In *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pages 1–10, Sept 2013.

[10] F. Hameed, L. Bauer, and J. Henkel. Architecting on-chip dram cache for simultaneous miss rate and latency reduction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(4):651–664, April 2016.

[11] F. Hameed, C. Menard, and J. Castrillon. Efficient stt-ram last-level-cache architecture to replace dram cache. In *Proceedings of the International Symposium on Memory Systems (MemSys'17)*, MEMSYS '17, pages 141–151, New York, NY, USA, Oct. 2017. ACM.

[12] G. HAMERLY. Simpoint 3.0 : Faster and more flexible program analysis. *Workshop on Modeling, Benchmarking and Simulation, 2005*, 2005.

[13] J. L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, Sept. 2006.

[14] C.-C. Huang and V. Nagarajan. Atcache: Reducing dram cache latency via a small sram tag cache. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, PACT '14, pages 51–60, New York, NY, USA, 2014. ACM.

[15] M. Jung, C. Weis, and N. Wehn. Dramsys: A flexible dram subsystem design space exploration framework. *IPSJ Transactions on System LSI Design Methodology*, 8:63–74, 2015.

[16] Y. Kim, W. Yang, and O. Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Comput. Archit. Lett.*, 15(1):45–49, Jan. 2016.

[17] G. Loh and M. D. Hill. Supporting very large dram caches with compound-access scheduling and missmap. *IEEE Micro*, 32(3):70–78, May 2012.

[18] G. H. Loh, S. Subramaniam, and Y. Xie. Zesto: A cycle-level simulator for highly detailed microarchitecture exploration. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 53–64, April 2009.

[19] C. Menard, J. Castrillón, M. Jung, and N. Wehn. System simulation with gem5 and systemc the keystone for full interoperability. 2017.

[20] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan. Enabling efficient and scalable hybrid memories using fine-granularity dram cache management. *IEEE Computer Architecture Letters*, 11(2):61–64, July 2012.

[21] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. Graphite: A distributed parallel simulator for multicores. In *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pages 1–12, Jan 2010.

[22] N. Muralimanohart and N. Balasubramonian, R.and Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 3–14, December 2007.

[23] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer. Hasim: Fpga-based high-detail multicore simulation using time-division multiplexing. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 406–417, Feb 2011.

[24] M. Poremba and Y. Xie. Nvmain: An architectural-level main memory simulator for emerging non-volatile memories. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 392–397, Aug 2012.

[25] M. Poremba, T. Zhang, and Y. Xie. Nvmain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems. *IEEE Computer Architecture Letters*, 14(2):140–143, July 2015.

[26] M. K. Qureshi and G. H. Loh. Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 235–246, Washington, DC, USA, 2012. IEEE Computer Society.

[27] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2: A cycle accurate memory system simulator. *IEEE Comput. Archit. Lett.*, 10(1):16–19, Jan. 2011.

[28] J. Stevens, P. Tschirhart, C. Mu-Tien, I. Bhati, P. Enns, J. Greensky, Z. Chisti, L. Shih-Lien, and B. Jacob. An integrated simulation infrastructure for the entire memory hierarchy: Cache , dram, nonvolati le memory, and disk. *Intel Technology Journal*, 17(1):184 – 200, 2013.

[29] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. Dramsim: A memory system simulator. *SIGARCH Comput. Archit. News*, 33(4):100–107, Nov. 2005.

[30] K. Wang, Y. Zhang, H. Wang, and X. Shen. Parallelization of ibm mambo system simulator in functional modes. *SIGOPS Oper. Syst. Rev.*, 42(1):71–76, Jan. 2008.

LXVI

Work in progress paper : A Fault Injection Platform for Early-Stage Reliability Assessment

A Fault Injection Platform for Early-Stage Reliability Assessment

Alexandre CHABOT
CEA, LIST - LAMIH
Software Reliability and Security Laboratory
P.C. 174, Gif-sur-Yvette, 91191, France
alexandre.chabot@cea.fr

Reda NOUACER
CEA, LIST
Software Reliability and Security Laboratory
P.C. 174, Gif-sur-Yvette, 91191, France
reda.nouacert@cea.fr

Ihsen ALOUANI
LAMIH
University of Valenciennes
Valenciennes, 59300, France
ihsen.alouani@univ-valenciennes.fr

Smail NIAR
LAMIH
University of Valenciennes
Valenciennes, 59300, France
smail.niar@univ-valenciennes.fr

ABSTRACT

With new integration rates, embedded systems sensitivity to environmental conditions has increased drastically. Thus, presenting reliable architectures is one of the major concerns for designers. During the development life-cycle of fault-tolerant computer systems, the dependability insurance is a complex and critical task. This work in progress paper presents a technique for reliability evaluation of embedded systems at early-stage by demonstrating the functionality and usage of simulation based fault injection. The proposed technique takes into account both the environmental conditions and the behavior of the application. Our results bring a flexible fault injection methodology and show its usability for reliability evaluation during simulation phase.

KEYWORDS

Fault injection, environmental conditions, application behavior, reliability, virtual prototype

ACM Reference Format:
Alexandre CHABOT, Reda NOUACER, Ihsen ALOUANI, and Smail NIAR. 2018. A Fault Injection Platform for Early-Stage Reliability Assessment. In *Proceedings of HIPEAC RAPIDO Conference (RAPIDO'18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Electronics are more and more present on our daily life. In fact, smart and connected systems are expected to be present practically everywhere in the future decade. This trend is made possible by the progress manufacturers and designers performed towards providing high performance platforms. However, this performance race has led to a degradation in embedded systems reliability. In fact, the reliability is a direct reflection of systems sensitivity to transient errors as well as to aging. Soft errors are caused by transient events that corrupt sequential and combinational elements and their impact is increasing dramatically in mainstream computing systems. Therefore, robustness techniques have been proposed to reach acceptable reliability levels. One of the hardware techniques consists of separating memory cells accessed by the same address to avoid multiple bit flips for the same stored word [10]. We can also cite the well-known error correcting code memory that is a computer data storage able to detect single bit flip. At software level, techniques such as task redundancy and the work of Huang et al. [14] on scheduling algorithms. Finally, CLEAR project [8] have compared techniques at different system layers.

The evaluation of embedded systems reliability is a complex task and different techniques are used during different development steps. Usually, reliability is linked to hardware manufacturing process and environmental conditions. However, having implemented robustness techniques at different system levels, it appears important to take it into account while evaluating the system reliability to avoid excessive design margins.

In this paper, we focus on transient errors and especially soft errors. To evaluate the system reliability, we propose a fault injection model that takes into account all parameters impacting reliability. In addition, our model takes into account the link between the application behavior and the fluctuation of the memory temperature, this link is exhibited in Section 3.1.2.

The rest of this work in progress paper is organized as follows. In the state of the art, Section 2, we discuss how fault injection associated to virtual platforms helps to ensure the correct functioning of critical systems. Then, in the third Section, we present our fault injection model. In the fourth Section, we discuss the evaluation of the proposed approach using UNISIM-VP [18] simulation environment and illustrate its flexibility by implementing some aspects of FIDES standard [11]. Finally, we present some preliminary experimental results.

2 STATE OF THE ART

In the domain of reliable embedded system design, many techniques have been considered to measure the robustness of an architecture. We can list here after the main ones:

- Analytical models: this technique can be used in the early phases of the design. It uses high level modeling of the system and approximations to determine the most critical functions in terms of reliability. We can cite the work done in SOPHIA [4]
- Prototyping: this method is often the last step of the development and the goal is to realize as less as prototypes as possible as it is a very expensive step [21].

RAPIDO'18, January 2018, Manchester, UK
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- Simulation: this technique is a trade-off between the analytical modeling and prototyping.

Since we will use simulation for reliability analysis, in the next we present a simulation based reliability assessment.

2.1 Simulation based reliability assessment

Simulation is used in the early design phase to ease the co-development of hardware and software [7]. It allows to identify mismatches that can occur during the integration phase. Early identifying mismatches helps in reducing the development cost as shown Figure 1. The co-development using simulation has two major advantages. First, the prototyping phase is reduced as less changes are needed. Later an issue is discovered more expensive is to correct it [21], and due to context of critical systems, the correction is mandatory. Second, the number of system errors discovered after commercialization is reduced. Because simulation allows to put the system in a given state, it is easier to test all possible execution paths and thus to entirely test given functionalities. Setting a system state is

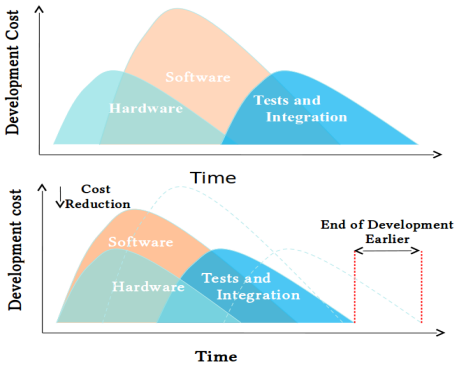


Figure 1: Development cost function of time using or not Virtual Platforms

harder using prototyping than using simulation [23]. Moreover, it costs time to repeat tests using prototyping, which is not the case in simulation.

Another advantage of simulation is the possibility to test deeply critical functions for critical systems. Simulation also reduces development costs, since the developed virtual platform can be reused in other projects using the same architecture. This is especially important for critical systems, because the developed platforms are supposed to be used a long time. Indeed, it is costly to certify an architecture for critical systems and that explain why the same platform is reused for different projects [20].

Nonetheless simulation has disadvantages. First, the virtual platform is an approximation of the system, it is not fully accurate. Those approximations are used to speed up the computation, a trade-off has to be made between the speed of computation and the accuracy of the simulation run. Second, even if the cost is reduced by reusing previously developed virtual platforms, it takes time at the beginning to develop a new virtual platform.

2.2 Fault Injection

Simulation is, in first way, used to test the correct functioning of the software onto a simulated hardware. The simulated hardware is supposed without defects and the environment is thus not taken into account.

In order to measure the reliability of a system, the classic simulation method is extended with a fault injection module. The fault injection module job is to simulate environmental perturbations that occur during the functioning of the system. In real world, the perturbation origin is either a particle that strikes the hardware and modifies the current state of the system or a hardware problem such as a latches fail. The disruption injected is materialized during simulation by a classic run perturbation. For example, it's possible to create a fault injection by modifying a bus value, flipping a bit in memory, or even by changing the next instruction to execute. Fault injection is used to verify the correct and time limited response of the system, even in the case of a perturbation. Originally used for reliable systems, the fault injection starts to be a tool in security especially in the context of hardware attacks such as explained in [22].

In a plethora of previous works, the fault injection module was based on a probabilistic model [24]. Indeed, the Poison's law with a λ constant representing the failure rate, governed the reliability of the system. The first model is thus purely random and the disruption of the system depends on two draws: one representing the disruption moment and the other representing the type of disruption. For the random injection, the equation representing the reliability law is Equation (1), where λ is the constant failure rate R is the reliability law, $MTTF$ is the mean time to failure and t is the time.

$$R(t) = \exp(-\lambda * t) \tag{1}$$

$$MTTF = 1/\lambda \tag{2}$$

Using this model implies to randomly test systems critical functions. However, functions have not the same criticality. In order to deeply validate a system with this approach it takes a long time as some runs does not test critical functions. To solve this issue and thus to gain time, a first approximation to the purely random approach was to focus injection moments on critical path of execution [24]. This second approach helps to reduce the time of simulation. However, a big random part is still present in the model. This random part is inefficient in our opinion and push us to work on fault injection model.

3 PROPOSED APPROACH

As explained earlier, fault injections used were mainly random. In addition, the model takes only into account the time and assume a constant failure rate that is far from the reality. Indeed, in real conditions, the application behavior may modify the failure rate of the system. Approximations can lead either to an over-protected architecture by robustness techniques that consumes a large part of execution time or power or an under-protected architecture. We aim to propose a more representative and configurable fault injection model. This new model can take into account many parameters which have the potential to impact the system reliability. As the application behavior is also taken into account, the model evolves during the simulation. In the idea to optimize robustness techniques

and simulation time, the model will also take into consideration the the proposition to focus onto critical paths developed in [24].

A new phenomenon ignored by previous models is taken into account in our approach: the single event multiple bit upsets [9]. In previous approach, a perturbation was directly linked to a single event for the hardware. For the memory, in a case of a single event single upset, a perturbation leads to a single bit flip. However, this approach is not valid anymore. Indeed, for real systems the majority of events, or strikes, lead to more than one bit flip. This may happen especially in SRAM-based memories for technologies under 40nm [9]. Up to our knowledge, this parameter is not yet considered by implemented fault injection model at the simulation level. It is an issue because of the under-evaluation of the system reliability that is linked to the lack of multiple bit upset consideration.

3.1 Qualitative Model

As said before, we consider problematical to continue with the random injection model as it is an approach far from the reality. To solve this problem, we present in this part an evolving fault injection model. This model has the characteristics to be more general and dynamic during simulation.

This model such as illustrated in Figure 2 is created around the answer of three questions:

(1) What kind of fault do we want to inject ?
(2) Where and when do we have to inject the fault ?
(3) What is the probability that the fault would happen ?

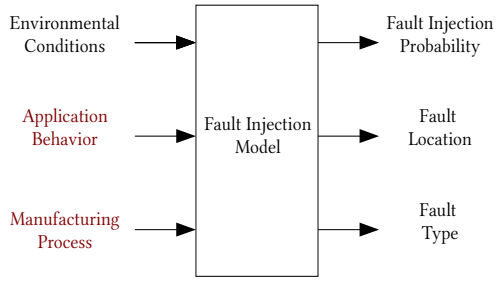


Figure 2: Schema of the Fault Injection Model

3.1.1 Fault Injected Type. In this part, we explain why the multiple bit upset is important to take into account and how we consider it into our fault injection model. With the technology scaling down, transistors followed (until 60nm) a decrease in term of sensitivity to particle strikes, however, this is not true anymore and below 60nm transistors are more and more sensitive to particle strikes [9]. In addition, we need to consider that the integration rate has increased during all this time, it means that for the same area more transistors are contained. Even if new transistors are not as sensible as their oldest, the number of soft error for the same area is constantly rising generation after generation.

Transistor miniaturization goes with the rise of multiple bit upset presence in the case of particle strikes. Multiple bit upset roads to become the majority of single event results as integration improves such as showed in Figure 3 in [9].

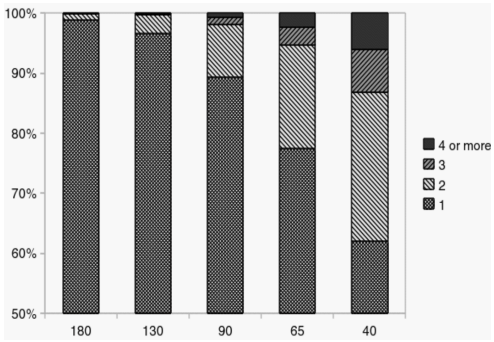


Figure 3: Multi-cell error percentages by technology nodes

During our investigation we did not find a fault injection model taking into account those multiple bit upsets. Regarding our research it seems important to both inject single upsets and multiple upsets. To achieve this wish, we investigate more deeply in the direction of multiple bit upsets and we identify the presence of more probable forms of multiple bit upsets, we call them injection patterns. In [19], we can clearly identify a higher probability for the memory studied that a 2-bit upsets involves two vertically connected cells than two horizontally connected cells. It exists thus a predominance of some patterns and we want to exploit this characteristics in our fault injection model. Moreover, depending on the technology the number of bit flipped during a particle strike evolves. We also take this characteristic into account in our fault injection model.

3.1.2 Location of the Injection. In previous approaches, the moment and the location of the injection were due to random number draws. However, this randomness does not fit in our wish to make the fault injection model more representative of the reality. It has been decided to take into account the behavior of the application to determine the location of the injection. In this part we are first going to validate and justify a statement needed to determine the location of injection: the frequency of access to a given area in memory impacts the sensitivity of this area to soft errors. Second, we will explain how we take it into account into our model to determine the location of the injection.

To justify the link between memory access and soft error, we first establish a link between the locality of memory access and the increase in temperature. Then we justify the link between the temperature of the memory and its sensitivity to particle strikes and thus to soft errors. The following logic resumes our path of thinking.

$$MemoryAccess \rightarrow Temperature \rightarrow SER$$

Memory access locality and Temperature relation. By accessing to a given area in memory (read or write operation), we modify the value of the current and by that way, we modify the power consumption of this area. By modifying bits in memory we thus heat up the area accessed as the temperature flows to close areas. This phenomenon is sometimes considered negligible [17]. However, it exists side channel attacks where the temperature modification due to operations is observed to permit attacks [15]. This convinces us to

consider this temperature modification as a tangible phenomenon. There is thus a direct link between the locality of memory access and the temperature modification.

Temperature and Soft Errors relation. We are going to justify the link between the memory temperature and the soft error rate. The work leaded by Bagatin [1] analyzes 5 commercial-available SRAMs, but only give conclusion on three of them due to manufacturing techniques that modify results for the two others and especially show unwanted behavior for critical memory. Indeed, the two outclassed showed single event latch-up during testing phase at room temperatures, those events are classically observed at the highest operating temperature. For the three device remaining, here follows a list of conclusion that have been raised:

(1) It exists a correlation between temperature and soft error rate
(2) The temperature increasing doesn't necessarily lead to an increase into the soft error rate. However, the manufacturing process must be taken into account to determine the shape of the SER function of the temperature.
(3) The supply voltage of the memory modifies the temperature impact onto the soft error rate. By reducing the supply voltage we reduce the impact of the temperature.

The variation of the soft error rate due to the temperature is modest but it is still valid to take it into account [16]. The main conclusion of the paper [1] says that there is a difference of up to 20% between the soft error at room temperature and temperature at the extrema of the device temperature range. Those 20% are not negligible fin the case of reliable systems. We thus establish a direct link between the temperature memory and the soft error rate of this memory.

Location of the Injection in our Model. We demonstrated above the link between the number of access to memory and the soft error. This relation is going to be taken into account in our model as our first step in the direction of taking into account the application behavior. More precisely, we weight the value of the fault injection moment by the analysis of the frequency of access to the memory (i.e. temporal locality). We implemented only a weighting of memory areas based on the locality, this weighting is introduced to ensure that all areas are evaluated from the reliability point of view.

3.1.3 Fault Probability. In our approach, the injection moment depends on the failure rate of the system. Such as showed in equation 1, the failure rate governs this equation. We didn't succeed into finding a better way to model the phenomena of soft errors. Nonetheless, the failure rate depends on environmental conditions and on manufacturing process. To model as well as possible this dependency on environment and the eventuality for a system to change of functioning conditions, we reuse the principle of life phases used by the FIDES group [11]. Those life phases associated to an environment and a duration are going to govern the modification of the failure rate depending on life phases and thus the failure rate would not be static anymore. This approach is in our opinion a way to answer a bigger number of critical system than a fixed failure rate as it is not static anymore. However, using only the failure rate to determine the fault injection timing is not feasible for complex system. Indeed, an entire coverage of all possible injections is impossible during simulation due to the time of computation needed to simulate all possible scenarios. Due to this assessment, it is rare to be complete from the test point of view. There is thus a need to reduce the number of simulations for a relevant exploration and some solutions have been explored.

A solution explored is to test only critical functions and not to depend on the failure rate. This approach leads to unexplored scenarios and does not test links between critical and not critical functions. Another approach is to test only most taken path of execution and use the failure rate in association. This second approach raises concerns also regarding unexplored scenario and evaluation of rarely called critical functions. We thus consider important to mix task criticality with failure rate to obtain a more pertinent fault injection model. We start our model reusing the random part used in the work of Huang et al. in [14], that considers all tasks sensible to perturbation and we aim to integrate tasks criticality in future works. The failure rate and tasks criticality are thus necessary to determine our fault injection probability. This failure rate is going to govern our injection decision such as shown Figure 4.

3.2 Integration of FIDES

Environmental conditions are most of the time considered one by one while studying their impact on reliability. We can cite papers like [13] and [25] which consider only the temperature or in the other case only the power consumption as environmental condition modifying the failure rate.

In order to provide a global methodology for evaluating reliability of critical systems during simulation, we consider in our model what has been done by the FIDES group and precisely their guide for reliability [11]. FIDES global electronic reliability engineering methodology guide is a global approach as it addresses all environmental conditions. This guide is composed by two major parts, only the first part regarding the reliability evaluation is used in our research. This methodology is used to compute the reliability of each electronic component taking into account manufacturing process and environment. Here is the list of environmental conditions that are taken into account in our model: life phases of the system, ambient temperature, temperature cycles, relative humidity, vibrations, saline pollution, environmental pollution, application pollution and chemical protection. Combination of those parameters allows to address different critical system environmental conditions. Adding to those environmental parameters, a base lambda in reference conditions is needed to compute the impact of the environment to it. This base lambda allows us to consider the manufacturing impact on the hardware reliability.

Based on FIDES model (3), physical and process contributions are separable for the failure rate called λ. We only keep the physical contribution to simulate environmental conditions.

$$\lambda = (\sum physical\ contributions * \prod process\ contributions) \quad (3)$$

The process contribution is based on the access frequency to a given area. Such as justified in part 3.1.3, the frequency of access to a given area in memory impacts the sensitivity to a particle strikes and thus the occurrence probability of a soft error in this place. We use this access frequency as a modifying coefficient of the failure rate. Therefore, the computed failure rate is different for each memory areas such as showed in (4), where f_i is the frequency of access to

the ith memory area and λ_i is the failure rate for the ith memory area.

$$\lambda_i = f_i * \lambda \quad (4)$$

Users are able to choose the number of memory areas that divides the global memory. Then the probability to obtain a multiple bit upset is computed and the memory modified regarding results obtained.

4 EXPERIMENTAL RESULTS

We tested our fault injection model using the MiBench benchmark suite [12], focusing onto the automotive subset. A mono-core system with an arm v7 simulator available on the UNISIM-VP website [6]. As mention in Section 4.1, this platform has been modified to allow the fault injection. We used this virtual platform to prove our concepts and to observe the impact of our implementation onto the simulation time. As it is a work in progress, the model showed Figure 4 is not yet fully tested. In this Section, we will show the impact of our module onto performance and the difference in terms of errors per type between single and multiple upsets.

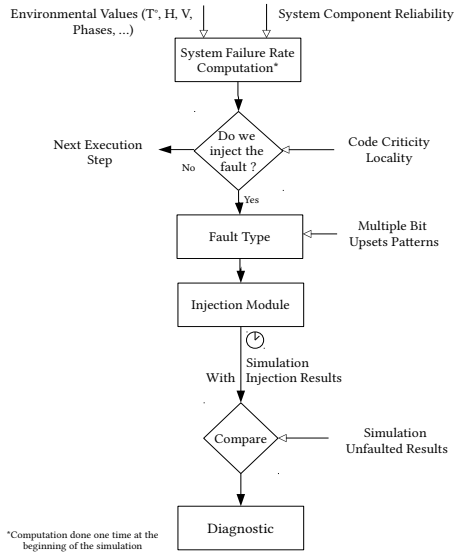


Figure 4: Fault Injection Strategy

4.1 UNISIM-VP

A large number of simulators such as [5] exist in the literature and have different characteristics. UNISIM-VP provides full system structural computer architecture simulators of electronic boards and System-on-Chip (SoC) using a processor instruction set interpreter. The whole software stack, consisting of the user programs, the operating system and its hardware drivers, is executed directly on the simulator. UNISIM-VP is a component-based software and is thus modular. Hardware components, written in the SystemC language [2], model the real target hardware components, such as CPU, memories, Input/Output, buses and specialized hardware blocks. Hardware components communicate with each other through SystemC TLM-2 [3] sockets that act like the pins of the real hardware. The service components are not directly related to pure computer architecture simulation. They allow initializing and driving of simulation. Services range from debuggers, loaders, monitors, host hardware abstraction layer and of course our fault injection module.

We use UNISIM-VP as our platform because of its transactional model that enable to build representative and efficient simulators. Moreover, its modular architecture enable re-usability and portability of our work to other simulation platforms.

4.2 Results

4.2.1 Performance. We provide results regarding the performance of our implementation. Unfortunately, as shown in Figure 5, the non-optimized version of our module has drastically impacted the simulation performance. This impact is explained by the functioning of our injection module. Indeed, it analyzes each memory access to determine which memory area has been accessed and how many times.

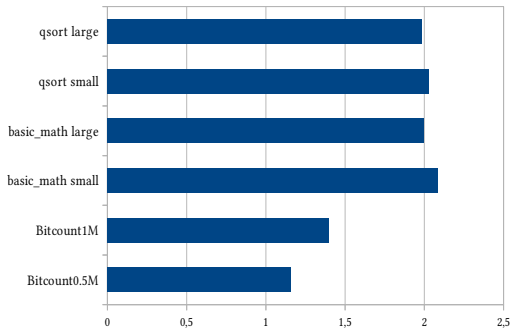


Figure 5: Multiplication of the simulation time for different tested application

Our first implementation has not shown a significant difference of the simulation computation time depending on multiple or single bit injection. As can be seen, the simulation time depends on the application interaction with the memory. Indeed, for the *Basic Math* and *Quick Sort* applications, the interaction with the memory is linear and thus the multiplication of the computation time due to our injection module remains the same even for larger inputs. However for the *Bit Count*, the memory interaction is not linear and depends on the input size. The injection module impact on computation time raises together with the number of elements to sort. We explain this difference by the strong interaction between the computation module and the memory in the case of the *Bit Count*.

4.2.2 Multiple Bit Upset Impact. We compare single bit injection and multiple bits injection. For all tested applications, no robustness techniques have been implemented and two bits injection have been used as the reference for multiple bit upset. First, not all multiple bit injection lead to a deviation from the golden run. Some faults are silent and thus do not affect the application behavior or the application result. Obtained errors, after fault injection, are separated in three categories: silent data corruption, behavioral

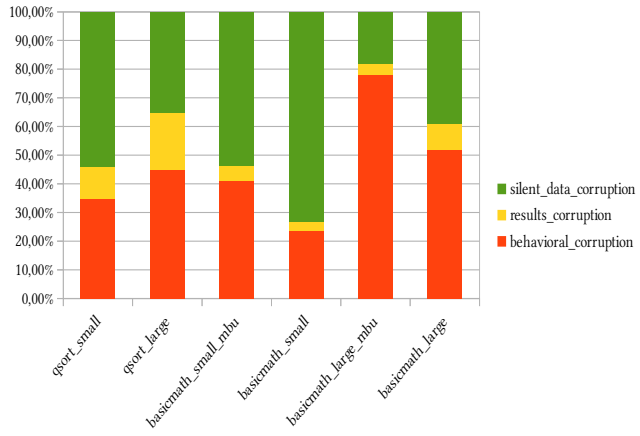


Figure 6: Number of results of each injection function of the application

corruption and result corruption. Figure 6, shows that when a multiple injection is used, less silent data corruption are obtained. The explication is that two scenarios can happen: first, two words in memory are modified; second, the same word is modified twice and thus is more different than if only one bit is modified. This justifies why multiple upsets leads to less silent data corruption.

5 CONCLUSION

In this paper, we highlighted issues regarding random injection and we propose an evolutionary fault injection model that takes into account environmental conditions, application behavior and the recent phenomena of multiple bit upsets. Regarding simulation results, the performance is a drawback to the first implementation of our approach, this is due to two factors. First our model is configurable and employable for different needs and is thus slower than a classic random number draw. Second, as it is the first implementation not a lot of optimization procedure have been applied. A first optimization would be to establish a analysis of the memory access during the golden run and to use it for all other simulations.

As a future work, we aim to improve our fault injection model by taking other environmental parameters into account and by speeding up our application behavior analysis. Moreover, we want to compare our methodology to existing ones to see the importance of this addition of precision. Finally, we wish to transfer our model to multi-core platform and to increase our number of use cases to observe the relevance of our work.

REFERENCES

[1] 2012. Temperature dependence of neutron-induced soft errors in {SRAMs}. *Microelectronics Reliability* 52, 1 (2012), 289 – 293. https://doi.org/10.1016/j.microrel.2011.08.011 2011 Reliability of Compound Semiconductors (ROCS) Workshop.

[2] Accellera. 2011. SystemC Standard Download page. (2011). http://www.accellera.org/downloads/standards/systemc

[3] John Aynsley. 2009. *OSCI TLM-2.0 language reference manual* (ja32 ed.). Open SystemC Initiative.

[4] Daniela Cancila, Francois Terrier, Fabien Belmonte, Hubert Dubois, Huascar Espinoza, SÃĺbastien GÃĺrard, and Arnaud Cuccuru. 2009. SOPHIA: a Modeling Language for Model-Based Safety Engineering. (2009).

[5] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulations. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 52:1–52:12.

[6] CEA. 2016. UNISIM Virtual Platforms. (2016). http://unisim-vp.org/site/index.html

[7] Jianjiang Ceng, Weihua Sheng, Jeronimo Castrillon, Anastasia Stulova, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. 2009. A high-level virtual platform for early MPSoC software development. In *CODES+ISSS '09: Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*. ACM, New York, NY, USA, 11–20. https://doi.org/10.1145/1629435.1629438

[8] Eric Cheng, Shahrzad Mirkhani, Lukasz Szafaryn, Chen-Yong Cher, Hyungmin Cho, Kevin Skadron, Mircea Stan, Klas Lilja, J.A. Abraham, Pradip Bose, and Subhasish Mitra. 2016. CLEAR: Cross-Layer Exploration for Architecting Resilience - Combining Hardware and Software Techniques to Tolerate Soft Errors in Processor Cores. (04 2016).

[9] Anand Dixit and Alan Wood. 2011. Impact of New Technology on Soft Error Rates. *Reliability Physics Symposim (IRPS)* (2011), 486–492.

[10] M. Ebrahimi, H. Asadi, R. Bishnoi, and M. B. Tahoori. 2016. Layout-Based Modeling and Mitigation of Multiple Event Transients. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 3 (March 2016), 367–379. https://doi.org/10.1109/TCAD.2015.2459053

[11] FIDES group. 2010. *Reliability Methodology for Electronic Systems.*

[12] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. [n. d.]. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. ([n. d.]).

[13] A. S. Hartman, D. E. Thomas, and B. H. Meyer. 2010. A case for lifetime-aware task mapping in embedded chip multiprocessors. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 145–154. https://doi.org/10.1145/1878961.1878987

[14] Jia Huang, Andreas Raabe, Kai Huang, Christian Buckl, and Alois Knoll. 2012. A framework for reliability-aware design exploration on MPSoC based systems. *Design Automation for Embedded Systems* 16, 4 (01 Nov 2012), 189–220. https://doi.org/10.1007/s10617-013-9105-6

[15] Michael Hutter and Jörn-Marc Schmidt. 2014. The Temperature Side Channel and Heating Fault Attacks. *IACR Cryptology ePrint Archive* 2014 (2014), 190. http://eprint.iacr.org/2014/190

[16] Y. Kagiyama, S. Okumura, K. Yanagida, S. Yoshimoto, Y. Nakata, S. Izumi, H. Kawaguchi, and M. Yoshimoto. 2012. Bit error rate estimation in SRAM considering temperature fluctuation. In *Thirteenth International Symposium on Quality Electronic Design (ISQED)*. 516–519. https://doi.org/10.1109/ISQED.2012.6187542

[17] M. Meterelliyoz, J. P. Kulkarni, and K. Roy. 2008. Thermal analysis of 8-T SRAM for nano-scaled technologies. In *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on*. 123–128. https://doi.org/10.1145/1393921.1393953

[18] Reda Nouacer, Gilles Mouchard, and Daniel Gracia-Perez. 2012. UNISIM Virtual Platforms. (01 2012). RAPIDO'12 - 4th Workshop on: Rapid Simulation and Performance Evaluation: Methods and Tools.

[19] D. Radaelli, H. Puchner, Skip Wong, and S. Daniel. 2005. Investigation of multi-bit upsets in a 150 nm technology SRAM device. *IEEE Transactions on Nuclear Science* 52, 6 (2005), 2433–2437.

[20] R. J. RodrÃŋguez and S. Punnekkat. 2014. Cost Optimisation in Certification of Software Product Lines. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. 509–514. https://doi.org/10.1109/ISSREW.2014.103

[21] Louis Scheffer, Luciano Lavagno, and Grant Martin. 2006. *EDA for IC System Design, Verification, and Testing (Electronic Design Automation for Integrated Circuits Handbook)*. CRC Press, Inc., Boca Raton, FL, USA.

[22] C. Shao, H. Li, G. Xu, and L. Xiong. 2014. Design for security test against fault injection attacks. *Electronics Letters* 50, 23 (2014), 1677–1678. https://doi.org/10.1049/el.2014.1666

[23] Charles Slayman. 2011. *JEDEC Standards on Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Errors.* Springer US, Boston, MA, 55–76. https://doi.org/10.1007/978-1-4419-6993-4_3

[24] B. Vedder. 2015. *Testing Safety-Critical Systems using Fault Injection and Property-Based Testing.* Halmstad. Licentiate dissertation.

[25] D. Zhu and H. Aydin. 2009. Reliability-Aware Energy Management for Periodic Real-Time Tasks. *IEEE Trans. Comput.* 58, 10 (Oct 2009), 1382–1397. https://doi.org/10.1109/TC.2009.56