View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by PORTO@iris (Publications Open Repository TOrino - Politecnico di Torino)

POLITECNICO DI TORINO
Repository ISTITUZIONALE

SystemC-AMS thermal modeling for the co-simulation of functional and extra-functional properties

Original
SystemC-AMS thermal modeling for the co-simulation of functional and extra-functional properties / Chen, Yukai; Vinco, Sara; Macii, Enrico; Poncino, Massimo. - In: ACM TRANSACTIONS ON DESIGN AUTOMATION OF ELECTRONIC SYSTEMS. - ISSN 1084-4309. - ELETTRONICO. - 24:4(2019), pp. 1-26.

Availability:
This version is available at: 11583/2716138 since: 2020-02-22T11:57:45Z

Publisher:
ACM

Published
DOI:10.1145/3267125

Terms of use:
openAccess

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

04 August 2020

SystemC-AMS Thermal Modeling for the Co-simulation of Functional and Extra-Functional Properties

YUKAI CHEN, Politecnico di Torino, Italy
SARA VINCO, Politecnico di Torino, Italy
ENRICO MACII, Politecnico di Torino, Italy
MASSIMO PONCINO, Politecnico di Torino, Italy

Temperature is a critical property of smart systems, due to its impact on reliability and to its inter-dependence with power consumption. Unfortunately, the current design flows evaluate thermal evolution ex-post, on offline power traces. This does not allow to consider temperature as a dimension in the design loop, and it misses all the complex inter-dependencies with design choices and power evolution. In this paper, by adopting the functional language SystemC-AMS, we propose a method to enable thermal/power/functional co-simulation. The system thermal model is built by using state-of-the-art circuit equivalent models, by exploiting the support for electrical linear networks intrinsic of SystemC-AMS. The experimental results will show that the choice of SystemC-AMS is a winning strategy for building a simultaneous simulation of multiple functional and extra-functional properties of a system. The generated code exposes an accuracy comparable to that of reference thermal simulator HotSpot. Additionally, the initial overhead due to the general purpose nature of SystemC-AMS is compensated by surprisingly high performance of transient simulation, with speedups as high as two orders of magnitude. The application of the proposed methodology to a set of benchmarks, used for the IEEE PATMOS design contest, will additionally prove the effectiveness of the SystemC-AMS thermal simulator.

CCS Concepts: • **Computing methodologies** → **Modeling and simulation**; • **Hardware** → *Temperature simulation and estimation*;

Additional Key Words and Phrases: Thermal analysis; Thermal estimation; Simulation; SystemC-AMS

ACM Reference Format:
Yukai Chen, Sara Vinco, Enrico Macii, and Massimo Poncino. 0. SystemC-AMS Thermal Modeling for the Co-simulation of Functional and Extra-Functional Properties. *ACM Trans. Des. Autom. Electron. Syst.* 0, 0, Article 39 (0), 27 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Assessment of extra-functional properties of an electronic system such as power consumption, reliability and thermal profile, is crucial to ensure correct operations; they should be considered at all design stages and concurrently, due to their complex interactions. While the modeling and simulation of such properties in isolation is a widely studied problem, the management of their inter-dependence and interaction at run time is only partially solved. In this scenario, temperature

Authors' addresses: Yukai Chen, Politecnico di Torino, Department of Control and Computer Engineering, Turin, Italy; Sara Vinco, Politecnico di Torino, Department of Control and Computer Engineering, Turin, Italy; Enrico Macii, Politecnico di Torino, Department of Control and Computer Engineering, Turin, Italy; Massimo Poncino, Politecnico di Torino, Department of Control and Computer Engineering, Turin, Italy.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 0 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
1084-4309/0/0-ART39 $15.00
https://doi.org/0000001.0000001

is one of the most critical properties, due to its strong correlation with functional operations (e.g., the workload) and power consumption. Temperature, in turn, impacts static (leakage) power consumption, and affects system reliability.

In the literature, a variety of solutions have been proposed for estimating temperature (both transient and steady-state), together with its mutual interaction with the other properties. The most straightforward solution is to evaluate each property with specific tools, and to use the output traces to integrate the various tools. The main problem of this solution is that the tools are executed in a predefined order, i.e., the thermal simulators are executed on traces produced offline by functional and power simulation tools [9, 30]. This prevents the possibility of evaluating the mutual influence of temperature and power. This limitation is usually overcome by the construction of co-simulation frameworks, that integrate dedicated simulators through complex synchronization and data exchange mechanisms [1, 3, 10]. However, co-simulation introduces a significant overhead, together with possible errors or timing misalignments due to different time scales or different models of computation (MoCs). Additionally, despite the simultaneous execution, co-simulation tends anyway to enforce a sequential estimation of the properties: functional and power simulation are used to generate the inputs for the thermal simulator, but there is no feedback on the corresponding temperature evolution. This implies that it is not possible to model any dependence on temperature (e.g., its impact on dynamic power consumption), and that thus more complex interactions between functionality, power and temperature can not be reproduced [1, 3, 17, 37].

Recently, Vinco et al. proved that this issue can be overcome by modeling both functionality and extra-functional properties with a single standard functional hardware description language [32], namely SystemC-AMS [12]. The reconciliation to a single simulation infrastructure and language allows indeed to simulate different aspects of a system in a single simulator instance, *i.e.*, a single run, and to reproduce at run time the mutual interactions while keeping the overhead low. *However, the authors described the syntactic and semantic rules needed to enable such homogeneous simulation, with no detailed instance of the proposed principles. This work presents how thermal simulation can be implemented in such a SystemC-AMS framework.*

The goal of this work is to define a methodology and an automatic tool that produce in output a SystemC-AMS implementation of the equivalent RC network, starting from input configuration information, i.e.*, the desired granularity of thermal estimation, the floorplan and a file modeling technology information. The RC network is constructed by using HotSpot as reference model [27]. The adoption of the SystemC-AMS language will allow both the straightforward implementation and simulation of the RC network (by exploiting the electrical linear network constructs and the underlying AMS solver) and to run models belonging to different domains (e.g., power and temperature) in a single simulation run, thus overcoming the aforementioned limitations of the state-of-the-art tools.*

The following are the specific contributions of this work:

- The integration of the SystemC-AMS thermal simulator in the framework proposed in [32], thus allowing the simultaneous simulation of power, reliability and functionality, and of their inter-dependencies with temperature evolution;
- A methodology to build the electrical circuit equivalent thermal models in SystemC-AMS. This paper builds upon [6], which implemented the RC network in SystemC-AMS limited to the "block" mode (*i.e.*, one temperature value per functional component). The novel contribution lies in the extension to different granularity of the thermal map (*i.e.*, grid level mode);
- A tool that automates the methodology by generating the SystemC-AMS RC network;
- An extensive experimental analysis, including:
  - The analysis of performance and accuracy of the SystemC-AMS thermal model on synthetic case studies, to prove the effectiveness of the generated code;

  - A characterization of the factors determining the performance of our method with respect to HotSpot, in particular those relative to the utilization factor of the floorplan and of the distribution of the values of power consumption;
  - The application to a benchmark suite used for the IEEE PATMOS design contest [13], to validate the proposed approach in a typical application scenario;
- The application of the proposed thermal model to the framework proposed in [32], to show a a simulation scenario including a dynamic mutual dependency between functionality, power models and thermal simulation.

Note that the goal of this paper is not to build yet another thermal simulator, but rather to allow the integration with power, functionality and reliability tools. Thus, we do not target better performance or accuracy w.r.t. HotSpot. Nonetheless, experimental results will confirm the effectiveness of SystemC-AMS at implementing a thermal simulator, in terms of both accuracy (the maximum error over all experiments is of 1%) and simulation time (with a maximum speedup between 15x and 60x for grid-level simulation, and of 280x for block level simulation). Additionally, the construction of an example of mutual dependency proves that the choice of SystemC-AMS is a winning strategy to gather a complete view of system evolution.

The paper is organized as follows. Section 2 provides the necessary background, *and Section 3 provides the motivation for this work.* Section 4 presents how equivalent circuit models can be modeled in SystemC-AMS. *Sections 5 and 6 provide the experimental setup and results, to analyze a set of simulation results corresponding to different simulation modes and benchmark sets, and prove the effectiveness of the proposed approach both in terms of accuracy and speedup.* Finally, Section 7 provides some concluding remarks.

2 BACKGROUND AND RELATED WORK

2.1 Thermal Modeling

When referring to silicon chips, temperature estimation amounts to solving the heat diffusion equation in 3D [22, 39]:

$$\nabla^2 T + \frac{\dot{q}}{k} = \frac{1}{\alpha}\frac{\partial T}{\partial t} \tag{1}$$

where T is the temperature, q is the heat flux (in W/m^2), k is thermal conductivity of the material (in $W/(mK)$), and $\alpha = \frac{k}{\rho c}$ is the thermal diffusivity, corresponding to the ratio between the thermal conductivity and volumetric heat capacity (density ρ times specific heat capacity c). $\nabla^2 T$ is the Laplacian of T and corresponds in 3D to $\frac{\partial T^2}{\partial x} + \frac{\partial T^2}{\partial y} + \frac{\partial T^2}{\partial z}$.

Equation 1 can be solved using numerical methods such as *Finite Difference Method (FDM), Finite Element Method (FEM), Model Order Reduction (MOR) or methods based on the Green function [15, 24, 35, 39].* Both FDM and FEM methods discretize the entire chip according to some granularity and construct a system of linear equations, thus handling complicated material structures with different thermal properties in different layers. Methods based on Green function, conversely, provide a semi-analytical approach that analyzes only the layers of interest. This reduces the problem size compared to FDM or FEM, but results in less accurate estimates due to the simplified modeling of the thermal problem.

Thermal analysis methods are further divided into two major classes, depending on the time dimension [39]. Steady state analysis determines the temperature distribution based on a power density distribution that does not change over time, by assuming either typical power values (*i.e.*, average power) or worst case scenarios (*i.e.*, maximum power consumption). This kind of analysis is thus useful to make preliminary design exploration, *e.g.*, to guide floorplanning or to make quick leakage estimations. On the contrary, transient analysis focuses on the temporal response to time

varying conditions, that is especially crucial to evaluate reliability effects. Accumulated heat can indeed determine peak temperatures much higher than the average values. Additionally, thermal stress strictly depends on thermal cycling and on the length of the peak temperature periods. As a result, transient simulation allows to gather a more complete estimation of the evolving circuit conditions, and allows to consider temperature as an additional design dimension.

2.2 Related Work on Thermal Simulation

A number of methods for solving Equation 1 rely on the well-known duality between thermal and electrical networks, *i.e.*, they represent heat flow as a current passing through a thermal resistance and leading to a voltage drop, corresponding to a temperature difference. Such methods rely on existing circuit-level simulators like SPICE to solve the steady-state or transient voltage of the nodes of the equivalent circuit [20, 36].

In the literature, a variety of tools have been proposed for estimating temperature (both transient and steady-state), with the goal of enhancing the design flow with increased reliability and knowledge of the underlying physical mechanisms, or of effectively managing modern many-core multiprocessors [16, 20, 21, 26, 27, 34, 36]. In the EDA and computer architecture community, the de-facto standard for thermal simulation is HotSpot, a tool based on a circuit-equivalent of a thermal network, which achieves a good trade-off between granularity and accuracy [27]. The equivalent circuit of the chip is built from a given floorplan and from the essential features of the die and of the package. HotSpot solves Equation 1 at each time step by using an adaptive solver of Runge-Kutta equations, based on a given trace of power dissipation values. It can model both steady-state and transient cases, and it supports two levels of granularity (*i.e.*, block-level and grid-level) with obvious tradeoff between accuracy and speed. Over time, different strategies have been proposed to reduce the overhead of the HotSpot equation solver. Some works implement context-specific optimizations, *e.g.*, by exploiting periodicity in the power trace [19], by using spatially- and temporally- adaptive techniques to reduce computation time [38], *or neural networks running on top of massively parallel architectures [31]*. Furthermore, the latest version of Hotspot introduces the adoption of advanced matrix manipulation libraries that exploit the architecture characteristics to parallelize operations like LU decomposition [14, 18, 40].

Although Hotspot can be considered as a state-of-the-art reference for thermal estimation, some other works have proposed variants of that scheme.

SESCTherm is a thermal modeling infrastructure based on finite-difference analysis [21]. Its underlying model is a mix of the execution modes of HotSpot (*i.e.*, block and grid mode): the division into regions is ruled by the involved materials and components, and irregular regions are further subdivided into quadrilateral regions. *Note that the goal of SESCTherm is not proposing a novel and sound thermal model, but to rather discuss the accuracy of thermal models,* e.g.*, by considering the impact of interconnect and mainboard PCB on the estimated thermal traces.*

*DTTEM uses a similar interface as HotSpot (*i.e.*, conductance and capacitance matrices) [28]*. The main difference lies in the construction of the input power consumption traces, that are built by segmenting cycle accurate traces according to a given time granularity and by calculating the average dynamic power consumption of each segment. This allows to determine different tradeoffs between accuracy and execution speed, and it simplifies the construction of the transient temperature traces.

HotSpot and the other approaches basically implement different strategies to speed up the solution of the differential heat diffusion equation *by optimizing either time sampling or analysis granularity*. However, they are relatively inefficient when performing thermal analysis for long simulation times, due to the occurrence of a large number of redundant computations intrinsic in the underlying solvers (as will be proven in the experimental section).

In this work we show that adopting SystemC-AMS allows to achieve the benefit of a concurrent simulation of functionality, power and temperature within a single simulation run, still guaranteeing good speed/accuracy tradeoffs with respect to HotSpot-inspired methods.

2.3 Introduction to SystemC-AMS

SystemC-AMS is the extension of SystemC for modelling analog and mixed-signal systems [12]. To cover a wide variety of domains, SystemC-AMS provides three different abstraction levels, supporting different communication styles and representations with respect to the physical domain.

Timed Data-Flow (TDF) models are scheduled statically by considering their producer-consumer dependencies in the discrete time domain. Each TDF module is characterized by a simulation time step, that is used by the TDF solver to insert timed activation events in the standard SystemC event queue. This ensures efficient computation, as it avoids any runtime dynamic event management.

Continuous time models can be modelled with two abstraction levels. *Linear Signal Flow* (LSF) supports the modelling of continuous time through a library of pre-defined non-conservative primitive modules (*e.g.*, integration, delay). The *Electrical Linear Network* (ELN) level models electrical networks through the instantiation of predefined primitives, *e.g.*, resistors or capacitors. Both ELN and LSF primitives are associated with linear equations, that are extended with energy conservation laws for ELN primitives.

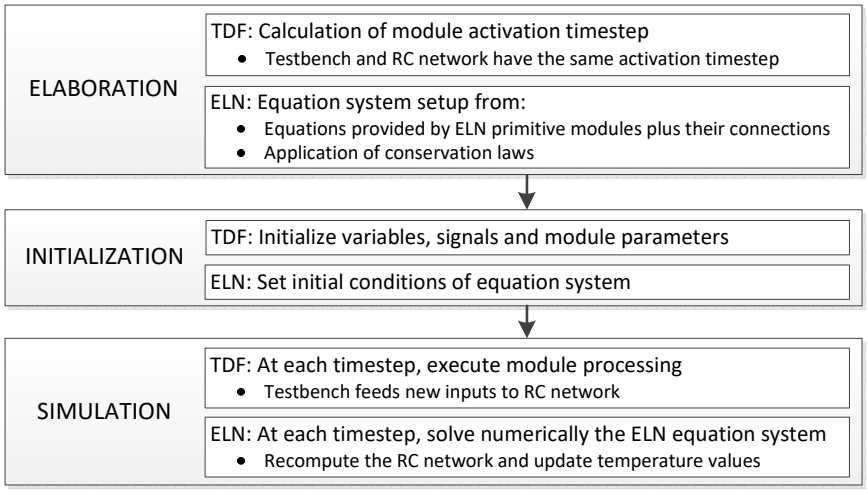


Fig. 1. *Execution flow of the SystemC-AMS simulation kernel.*

SystemC-AMS simulation is managed by an internal simulation kernel, whose execution flow is highlighet in Figure 1. At elaboration time, *SystemC-AMS builds module hierarchy and the data structures necessary to handle simulation. The simulation kernel determines also the activation condition of each module, i.e., the activation time step of TDF modules, and the input signals for SystemC processes and for ELN and LSF modules. This allows to setup the scheduler internal data structures.*

As a next step, the simulation kernel builds the equation system generated by ELN and LSF equations, as derived from the instantiated primitive modules. ELN equations are additionally extended with the application of energy conservation laws. The subsequent phase is the initialization phase, *that is used to setup module parameters and initialize system components (for TDF) and to setup the equation initial conditions (for ELN and LSF).*

The core of the simulation is then the simulation phase. *At any simulation cycle, the SystemC-AMS simulation kernel determines the next event to be processed and the corresponding execution queue.*

Events can be timing events (e.g., activating TDF modules) or value changes for ports and signals. Execution of ELN and LSF modules activates a linear DAE (Differential Algebraic Equation) solver, that numerically solves the corresponding equation system over time. The DAE solver uses lightweight numerical methods, i.e., *backward Euler and trapezoidal methods, combined with optimization techniques,* e.g., *LU decomposition and Woodbury formulas [8, 25].*

3 EXTRA-FUNCTIONAL PROPERTY SIMULATION IN SYSTEMC-AMS

The solution proposed by Vinco et al. in [32] formalized the reconciliation of functional and extra-functional properties to a single language by proposing a structured co-simulation. Its main characteristic is that it handles each property as a separate view of the system (called *layer*), thus managing information related to each property independently (as reported in Figure 2). The adoption of a single language for all properties allows on the other hand to simulate all layers concurrently, in a single run, and to intuitively reproduce mutual interactions between layers, without incurring in additional overheads. To achieve this result, the framework relies on SystemC-AMS, that proved to effectively support extra-functional domains, thanks to its modularity, flexibility, and the support of multiple MoCs [4, 6, 33]. Each layer of the framework is additionally constrained to a single underlying architecture, whose central element is the *layer-specific bus* (to reflect the typical functional architecture). Each bus in in charge of carrying the information flow related to its layer, with no necessary mapping onto actual system components. The way in which information carried by the bus is aggregated determines the simulation semantics of each layer.

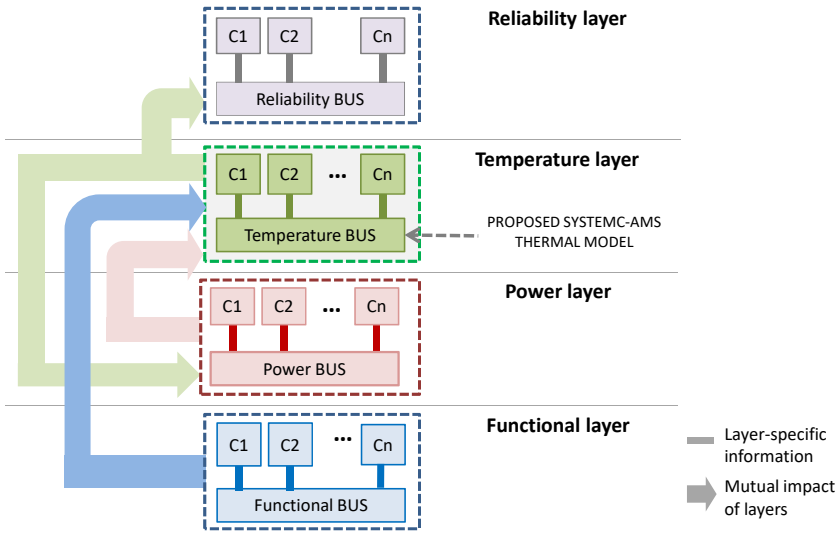


Fig. 2. Application of the framework proposed in [32] to the modeling of the mutual impact of temperature on functionality, power consumption and reliability.

Temperature is one of the native layers of such framework, due to its strong correlation with both functionality, power consumption and reliability. The bold arrows in Figure 2 represent the mutual dependency between layers, restricted to the interactions involving the temperature layer (the original figure with all framework details can be found in Figure 1 of [32]). *Despite of giving a formalized structure of the framework, Vinco et al. did not provide a detailed instance of the proposed principles. This work aims at filling the temperature layer of their framework with a state-of-the-art circuit equivalent model of the thermal evolution, that can thus be easily simulated together with all other system properties.*

4 A SYSTEMC-AMS THERMAL SIMULATOR FOR SOC

The construction of the SystemC-AMS thermal model relies on electrical circuit equivalent models as in HotSpot, exploiting SystemC-AMS ELN MoC, which explicitly supports the modeling and simulation of electrical elements. It is worth emphasizing that the novelty of this work does not lie in the thermal model itself, which closely follows the one used by HotSpot [27], but rather in the code generation process, which builds a simulatable model in which the electrical circuit components are explicitly instantiated. HotSpot provides in fact a stand-alone tool that explicitly solves circuit equations modeled as matrices. Conversely, the proposed approach exploits the native support of SystemC-AMS for electric network primitives to map the RC network elements one-to-one to SystemC-AMS constructs (*e.g.*, resistors and capacitors); the corresponding equations are then automatically derived and solved by the SystemC-AMS internal solver. The current paper extends the preliminary analysis carried out in [6] and provides support of *all* simulation modes featured by HotSpot, *i.e.*, both transient and steady state simulation, and different spatial granularities, *i.e.*, both grid level and block level modes.

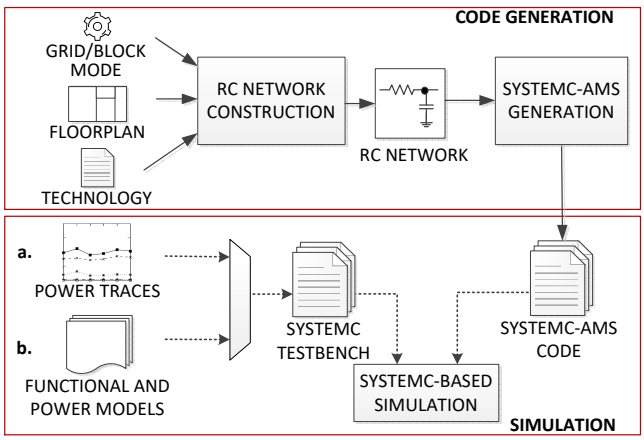


Fig. 3. Proposed methodology for the construction of the SystemC-AMS thermal simulator.

The flow of the proposed methodology consists of the following phases (Figure 3):

- *Construction of the RC network*, by reproducing the method used by HotSpot (Section 4.1) using chip floorplan and technology data;
- *SystemC-AMS code generation*, achieved by mapping the RC network elements to SystemC-AMS ELN primitives (Section 4.2);
- *Stimuli generation* through the construction of a dedicated testbench, implemented either as (a) a trace of power consumption values (á la HotSpot), or (b) by simulating the thermal model concurrently with functional and power execution (Section 4.3);
- *Simulation of the RC network* by using the SystemC-AMS simulation kernel (Section 4.4).

4.1 Construction of the RC Network

The algorithm to construct the RC network reproduces the method followed by HotSpot, by supporting both steady-state and transient simulation, and both grid- and block-level modes.

The construction of the RC network uses the input information, which include (i) the chip floorplan, necessary to determine the adjacency among components, (ii) technology information (*e.g.*, number of layers, materials, thermal characteristics), and (iii) the granularity of the thermal

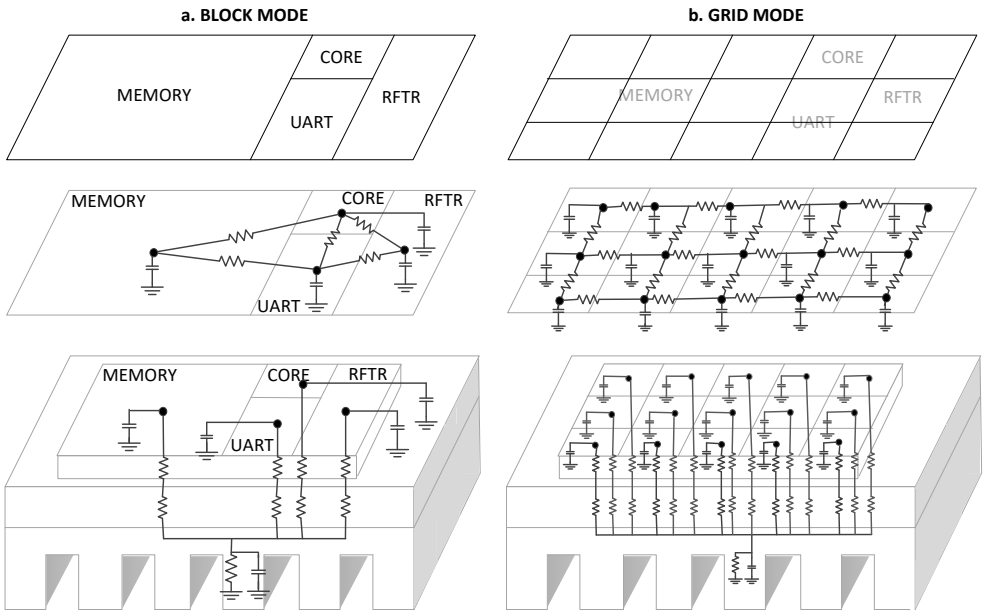


Fig. 4. Example of electrical circuit equivalent model. The number of nodes of the RC network depends on the desired level of accuracy (top), *i.e.*, block level mode (a) or grid level mode (b). The RC network is decomposed in a lateral model (middle, representing heat transfers that occur within a layer) and a vertical model (bottom, that captures the heat spread across the package layers).

map (block- vs. grid-). Figure 4 exemplifies the application to a case study consisting of four blocks (representing a system with a core, a memory block, a RF transceiver and a UART device) [7].

4.1.1 Identification of Network Nodes. As already mentioned, there are two different granularity levels for the thermal simulation (block and grid), which imply different two generation processes. concept from the perspective of the generation of the electrical network.

The block model and the grid model provide two different granularities to thermal analysis. The block mode is suitable for scenarios where preserving the mapping of thermal nodes to functional components is fundamental, e.g., to determine the reliability of a component, or the mutual impact of its power consumption and temperature. Vice versa, the grid version is suitable for analysis that work at lower levels of abstraction, e.g., to influence the synthesis process with information on possible hotspots, thus requiring a finer granularity and to keep the focus is at chip level, rather than on the functional components. Thus, even if the underlying RC network model is the same, the two simulation modes differ in terms of simulation characteristics.

The *block mode* associates one temperature value to each functional component. To this extent, the center temperature of the component is considered as the temperature of the component as a whole, and each functional component is represented by a single network node (top of Figure 4-(a)).

The *grid mode* allows a finer granularity by dividing the floorplan into a regular grid of cells having all the same size, with no dependency from the actual functionality of the components (top of Figure 4-(b)). The division in cells is indeed based solely on the area of the floorplan and on the desired granularity and degree of accuracy, as determined by the user [11]. In this mode, the current flowing across each node and its voltage represent the portion of power dissipation generated by the corresponding area of the floorplan and the temperature of the same area, respectively.

Additional nodes are used in both modes to represent the underlying package layers.

4.1.2 Instantiation of Network Elements. The RC network consists of two integrated models [27]. The *lateral model* (middle of Figure 4) captures heat diffusion between adjacent nodes within a layer, while the *vertical model* (bottom) captures the heat flow from the die through the package and eventually into the air. Heat transfer flow between adjacent nodes is represented by a resistor, whose thermal resistance is proportional to the thickness of the material and inversely proportional to the cross-sectional area and to the thermal conductivity. If transient simulation is required, capacitors are connected to each node, to capture the delay before a change in power determines a change in temperature. Thermal capacitance is proportional to both thickness and area, and it depends on the thermal capacitance per unit volume.

Figure 4 clearly shows that the chosen generation mode (block vs. grid) significantly affects the accuracy/complexity tradeoff of the thermal simulation; as the grid element size decreases we have more grid elements and the number of RC network elements grow accordingly.

Given the system-level perspective of our work, phenomena more related to the circuit physical characteristics (*e.g.*, wiring thermal contribution, or 3D modeling) are not supported in the current version of the methodology. Accuracy of the model could additionally be improved by considering the thermal dependence of conductivity and conductance from temperature. Even if including this dependence in the model would be straightforward, HotSpot itself considers this as a marginal contribution and thus it does not support it in the current version [11].

4.2 SystemC-AMS Code Generation

The second step is the implementation of the extracted RC network in SystemC-AMS. Figure 5 exemplifies the implementation of the lateral model depicted in Figure 4.a in SystemC-AMS[1].

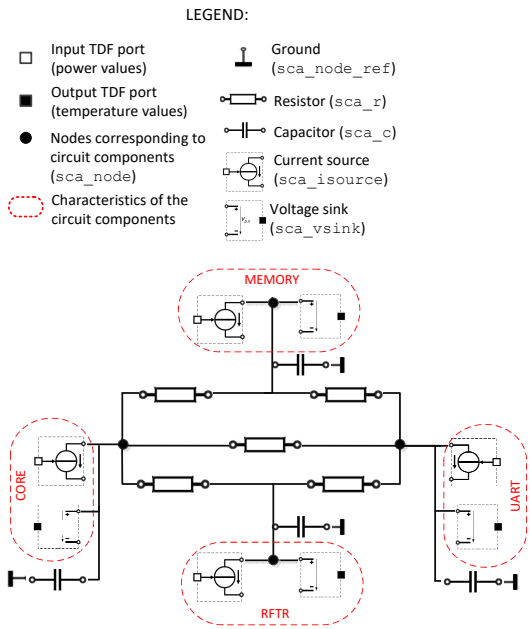
4.2.1 Interface Modeling. The electrical circuit equivalent model is instantiated as a single SystemC module (`SC_MODULE`, line 1) that encapsulates the entire RC network. The interface of the module consists of two ports for each component (for the block mode) or grid cell (for the grid mode): one input port is used to collect the evolution of power consumption over time, whereas the output port conveys the corresponding temperature value.

The flexibility of SystemC-AMS allows to decouple the semantics of the interface from the semantics of the actual behavior implementation. For this reason, the abstraction level adopted for ports is TDF, that determines a fixed timestep at which input ports are read, RC network is evaluated and output ports are updated. This reflects the behavior of HotSpot, that assumes that the power traces contain samples collected at fixed time steps. Ports are thus declared as `sca_tdf::sca_in` and `sca_tdf::sca_out` ports of type double (lines 2–3).

4.2.2 Module Implementation. The body of the `SC_MODULE` includes the implementation of the RC network. This is achieved by adopting the ELN level of abstraction to reproduce the model computed as in Section 4.1. The construction of the SystemC-AMS thermal model is thus a one-to-one mapping of circuit elements into SystemC-AMS primitives.

Each circuit node is implemented as a SystemC-AMS ELN node (`sca_node`, lines 6–7), but for the ground element, which is represented with an ad-hoc primitive (`sca_node_ref`, line 5). Resistors are mapped to instances of the `sca_r` primitive, which represents SystemC-AMS resistors (line 12). The resistance value is extracted from the resistance matrix computed in the previous step; for instance, Figure 5 shows that the resistance modeling the heat flow between the core and the memory is 215.48Ω (lines 27–30). Capacitors are mapped to an instance of the `sca_c`

[1]Note that this constitutes only a subset of the complete RC network. Symbols adopted for the primitives are as standardized in the SystemC-AMS standard [12].




```
1.  SC_MODULE (thermal_network){
2.    sca_tdf::sca_in<double> p_CORE ;
3.    sca_tdf::sca_out<double> t_CORE ;
4.    ...
5.    sca_eln::sca_node_ref gnd;
6.    sca_eln::sca_node node_CORE;
7.    sca_eln::sca_node node_MEMORY;
8.    ...
9.    sca_eln::sca_tdf::sca_isource* iCORE;
10.   sca_eln::sca_tdf::sca_vsink* vCORE;
11.   sca_eln::sca_c* c_CORE;
12.   sca_eln::sca_r* r_MEM_CORE;
13.  ...

14. SC_CTOR(thermal_network){
15.   i_CORE = new sca_eln::sca_tdf::sca_isource("i_CORE");
16.   i_CORE->p(gnd);
17.   i_CORE->n(node_CORE);
18.   i_CORE->inp(p_CORE);

19.   v_CORE = new sca_eln::sca_tdf::sca_vsink("v_CORE");
20.   v_CORE->p(node_CORE);
21.   v_CORE->n(gnd);
22.   v_CORE->outp(t_CORE);

23.   c_CORE = new sca_eln::sca_c("c_CORE");
24.   c_CORE->p(node_CORE);
25.   c_CORE->n(gnd);
26.   c_CORE->value=0.00008741;

27.   r_MEM_CORE= new sca_eln::sca_r("r_MEM_CORE");
28.   r_MEM_CORE->p(node_MEMORY);
29.   r_MEM_CORE->n(node_CORE);
30.   r_MEM_CORE->value = 215.48122572;
31.   ...
32. } };
```

Fig. 5. Example of implementation of the block level mode lateral model of Figure 4 in SystemC-AMS: implementation of the RC network modeling heat transfers within layers with ELN primitives (left) and corresponding excerpt of SystemC-AMS code (right).

primitive, representing SystemC-AMS capacitors (line 11). The capacitance value is extracted from the computed capacitance array; in the figure, the core capacitance is $8.741 \cdot 10^{-5}$F (lines 23–26).

The input power ports are connected to the ELN circuit via current source primitives, that transform a numerical value into a current (`sca_isource` primitive, lines 9 and 15–18). In the pictorial representation in Figure 5, TDF ports are represented by the white square terminals of the primitive blocks. Conversely, the temperature is extracted through a voltage sink, that extrapolates a voltage value and makes it available on the output ports. Voltage sinks are represented with instances of the `sca_vsink` primitive, whose output terminal is connected to the corresponding output temperature port (the black square terminals, lines 10 and 19–22).

4.3 Stimuli Generation

The final step is the generation of input stimuli. The SystemC-AMS thermal simulator is connected to a testbench module having an interface complementary to the that of the thermal network: power ports are in output, and temperature ports are in input. The testbench generates stimuli over time for the thermal simulator, by adopting the two possible strategies shown in Figure 3.

4.3.1 Static Stimuli Generation. The first option is to load power consumption traces from files previously generated by a power simulator (option *a* in Figure 3), and to dump the estimated temperature values to a thermal trace file for future elaborations. This corresponds to the typical operations of existing thermal simulators, including HotSpot. In block-level mode, a power trace per component is needed, and the relative power values can be either measured or estimated by a

power simulator. In the grid-level mode, which requires a power trace per each cell, the process is slightly more elaborated. Since power is associated to components and not to grid points, the power consumption of grid cells can be estimated by dividing the power consumption of functional blocks over the cells covered by that component, and proportionally to their respective occupied area.

4.3.2 Dynamic Stimuli Generation. The second option for the generation of input stimuli consists of simulating the RC network using power values *dynamically generated* with functional simulation (option *b* in Figure 3). In this scenario, the testbench wraps SystemC processes implementing the power models, or functional models enriched with power information [2, 23]. This approach is made possible by using of a functional language for thermal simulation. A major advantage is that power information is generated dynamically, depending on the evolution of functionality, and that the mutual effects of power and temperature can be reproduced at runtime [32].

An example of this strategy is sketched in Figure 6 for the core component of Figure 4. The testbench includes the functional model of the component, that uses a `pause` signal to put part of the functional processes to idle on demand. The `pause` signal is used also to drive a power model of the device, implemented as a state machine that dynamically updates the power values depending on the core power mode. The produced power value is written to the `p_CORE` output port, that is connected to the RC network in Figure 4. This allows one to adjust the power demand to the core functional evolution at run time, and to derive the corresponding temperature evolution from dynamically generated stimuli (`t_CORE` port).

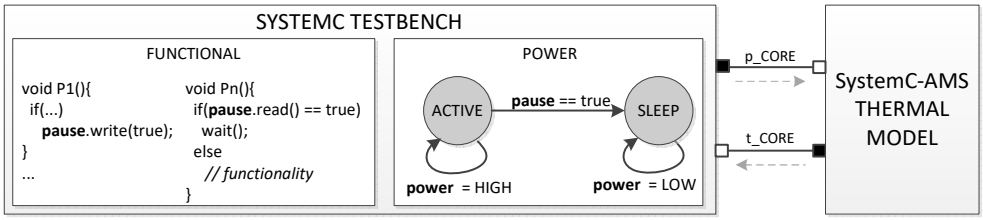


Fig. 6. SystemC testbench enabling concurrent power and thermal simulation.

A special case of this category is the insertion of the thermal simulator in the framework proposed in [32] (presented in Section 3). To reach this result, the generated SystemC-AMS thermal network is accommodated by the thermal bus, that acts as a centralized thermal simulator (as depicted in Figure 10). At simulation time, each component forwards its power consumption over time to the temperature bus, that in turn provides the corresponding evolution of the component temperature.

It is worth emphasizing that the dynamic stimuli generation scenario represents an important novelty with respect to the current state-of-the-art. Concurrent simulation is in fact achieved by simulating all aspects in a single simulation run and by adopting a single simulation kernel, *i.e.*, SystemC. Comparing this solution with approaches such as [1, 3, 10], we avoid the overhead of co-simulation, thus achieving faster simulation and an enhanced guarantee of correctness [32].

4.4 SystemC AMS simulation

This section presents how the generated code is managed and simulated by the SystemC-AMS simulation kernel, with respect to the standard execution flow presented in Section 2.3 and Figure 1.

At elaboration time, the timestep associated to the TDF testbench and to the TDF ports of the RC network are used to determine how often the thermal simulator should be re-evaluated. This corresponds to the semantics of HotSpot, which estimates the evolution of temperature at fixed time points. Then,

the ELN primitives implementing the RC network (instantiated similarly to Figure 5) are used to build the ELN equation system, that is enriched with the application of energy conservation laws.

*During the simulation phase, activation events coincide with the advancement of time: time is advanced by one TDF time step, thus triggering the execution of the TDF modules (*i.e.*, of the testbench) and recomputing the values of the RC network on the new inputs. Note that when the thermal simulator is executed in more complex scenarios (*e.g.*, the one in Figure 6), the thermal TDF time step may co-exist with other events, that will be used by the solver to activate and synchronize all system modules.*

As anticipated in Section 2.3, the equation system is solved numerically by the linear DAE solver. The chosen methods (i.e., backward Euler and trapezoidal methods) lead to optimal simulation performance in other contexts, still preserving a good level of accuracy with respect to more sophisticated tools [6, 33]. The choice of lightweight methods constitutes a difference with respect to HotSpot, that adopts more complex methods to achieve higher accuracy and stability, e.g., *4th order Runge-Kutta (that goes through a number of Euler-style steps to match a Taylor series expansion) [8]. Given that both tools solve the same RC network, any difference in terms of accuracy or simulation time will reside in the choice of different DAE solvers. This will be the object of the experimental analysis.*

5 EXPERIMENTAL SETUP

This section presents the setup of the experimental analysis, by focusing on adopted tool and simulator versions, on the automation of the proposed methodology, and on the adopted benchmarks.

5.1 Simulation Setup

Experiments have been run by using SystemC 2.3.1, SystemC-AMS 2.1 and HotSpot 6.0, on an Intel Xeon 2.40GHz CPU (8 cores, 16 threads each) with 128GB RAM, running the CentOS 6.7 Linux operating system. Simulation times are calculated as the average over 100 simulations.

As mentioned earlier, recent versions of HotSpot adopt accelerators for smart management of matrices. For the experimental analysis, we considered two of the available accelerators. SuperLU is an open source library for direct solution of sparse systems of linear equations, and it is adopted by HotSpot only for steady simulation and in grid mode [18, 40]. For transient simulation, HotSpot 6.0 adopts a number of architecture-specific accelerators, including the Intel Math Kernel Library (MKL) [14], that automatically partitions matrix operations onto a number of pthreads.

5.2 Automatic RC network generation

The construction of the SystemC-AMS RC network has been automated in a C++ tool, that takes in input the desired simulation mode (*i.e.*, grid or block level, and steady state or transient) and two files, modeling floorplan and technology information, respectively. The floorplan file contains a line per component, and for each component it lists the width, height, and (x,y) coordinates of the lower-left corner in meters. The technology information file follows the format defined by HotSpot, and it lists the configuration parameters as command line options. The whole generation process of RC network is automatic, and it saves the generated code to a SystemC file. As a by-product, the tool also generates the interface of the testbench module to be connected to the RC network and used in the simulation, while its implementation is left to the designer.

5.3 Benchmarks

We used five benchmarks of different sizes and complexity. Benchmark *1* is the example in Figure 4: we used this relatively simple floorplan in order to easily investigate the internal simulation mechanisms between SystemC-AMS and Hotspot. Benchmark *2* and *3* are the typical case studies of HotSpot: an Alpha 21364 microprocessor and a POWER4-like microprocessor. The simulation results of these two microprocessors can provide a real thermal simulation, compared to the

synthetic benchmark *1*. We additionally created two synthetic benchmarks (*4* and *5*), by replicating the components of benchmarks 2 and 3 (and the corresponding power consumption traces) a number of times, such as replicating cores and caches in the floorplan. We selected them in order to analyse effectiveness and scalability of the proposed simulation approach. All floorplans have been derived by using HotFloorplan, *i.e.*, the thermal-aware floor-planning tool included in the HotSpot distribution.

To ensure a fair comparison, both HotSpot and our tool have always been run with the same time step (of 1ms) and fed with the same power traces, floorplans and technology files. Table 1 summarizes the adopted technology values. Air is set to a fixed ambient temperature of 40^oC. Convection capacitance with respect to ambient is $140.4\frac{J}{K}$, and convection resistance is $0.1\ \frac{K}{W}$.

	Die	TIM	Heat-spreader	Heat-sink
Thickness [m]	1.5e-4	2.0e-5	1.0e-3	6.9e-3
Thermal conductivity [$\frac{W}{m \cdot K}$]	100.0	4.000	400.0	400.0

Table 1. Technology parameters.

6 EXPERIMENTAL RESULTS

In this section we demonstrate the effectiveness of the proposed SystemC-AMS thermal simulator by comparing simulation accuracy and performance with respect to HotSpot. Given that the underlying model is identical for the two simulators, it is thus crucial to apply an extensive experimental analysis to understand the behavior of the generated code and its main stregths and weaknesses w.r.t. HotSpot.

The experiments are organized as follows. Sections 6.1 and 6.2 focus on the block-level mode and on the grid-level mode, respectively, to evaluate accuracy and performance of the generated code. Section 6.3 compares the characteristics of the two modes on a benchmark. Section 6.4 applies the experimental analysis to a set of benchmarks provided by the IEEE PATMOS design contest [13], to validate the proposed approach in a typical application scenario. Section 6.5 proposes a case study where the thermal simulation is run in parallel with functional and power models, by adopting the framework in [32]. Finally, Section 6.6 discusses the main findings and outcomes of the experimental analysis.

6.1 Block-Level Mode

Benchmark	Components	Nodes	Rs (#)	Cs (#)	Floorplan density (%)
1	4	28	64	28	98.91
2	18	84	288	84	98.22
3	30	132	442	132	97.10
4	40	172	586	172	95.36
5	86	356	1,206	356	93.62

Table 2. Characteristics of the computed RC networks for the block level mode.

Table 2 summarizes the main characteristics of the RC networks of the five benchmarks, in terms of number of nodes, resistors, and capacitors. The floorplan density column will be commented later on in the paper. The network built by our approach for each benchmark is identical to the one built by HotSpot, since the two network construction algorithms are similar. Therefore, both SystemC-AMS and HotSpot solve the same equations; any difference in execution times and in accuracy is thus due only to the underlying solvers, and not to differences in the networks.

6.1.1 RC Network Initialization. Table 3 reports the time required to build the RC network for HotSpot and SystemC-AMS. Time is obviously correlated with the number of component, since in order to determine the value of any resistor or capacitor, the algorithm must consider the

characteristics of the involved blocks and layers, together with the neighbouring relations between components (*e.g.*, shared length of two neighbouring blocks). Similarly, initialization time is larger for transient simulation, since the RC network includes also one capacitor per network node.

Benchmark	Steady state simulation [ms]		Transient Simulation [ms]	
	SystemC-AMS	HotSpot	SystemC-AMS	HotSpot
1	0.558	0.318	0.655	0.341
2	1.840	0.705	2.528	0.751
3	3.262	1.633	3.959	1.802
4	4.951	2.528	5.896	2.997
5	14.835	10.067	16.084	13.711

Table 3. Comparison of RC Network Initialization Time for the block level mode.

The construction of the RC network in HotSpot and SystemC-AMS implies different sequences of operations; in HotSpot, the values of thermal resistances and capacitances are directly stored into matrices to be used by the circuit solver. Conversely, SystemC-AMS needs to explicitly instantiate resistances and capacitances as connected ELN instances in a source file (see Figure 5), which is compiled and used by the SystemC-AMS solver to build the underlying system matrices. The SystemC-AMS flow has therefore an intrinsically higher overhead. In spite of that, for such small numbers of blocks, the initialization times for the two simulators are comparable.

6.1.2 Steady-State Simulation. Table 4 reports the simulation time for the steady-state mode of SystemC-AMS code and HotSpot. Note that no accelerator is available for HotSpot in this configuration. Steady state stimulation is very efficient for both simulators, at it requires the solution of just one, relatively simple RC network (maximum 1,200 resistors, no capacitors). As Column *speedup* shows, simulation times are comparable, although SystemC-AMS tends to scale better with network sizes (from 1.1x for the smallest network to 2.3x for the largest one).

Benchmark	HotSpot [ms]	SystemC-AMS	
		[ms]	speedup
1	10.476	9.563	1.10x
2	45.079	40.024	1.13x
3	95.343	84.365	1.13x
4	121.748	99.981	1.22x
5	565.041	246.014	2.30x

Table 4. Comparison of Steady-State Simulation Time for the block level mode.

6.1.3 Transient Simulation. Table 5 reports the performance of SystemC-AMS and HotSpot (with and without the MKL accelerator) with transient simulation. With transient simulation the advantage of the SystemC-AMS implementation becomes evident. The speedup again improves with the size of the circuit, reaching a remarkable 279x for the largest benchmark. The speedup is mainly due to the differences between the solvers. As briefly mentioned in Section 4.4, the SystemC-AMS solver uses lightweight numerical methods, *i.e.*, backward Euler and trapezoidal methods, thus accepting to potentially sacrifice accuracy for the sake of simulation speed. Further optimization techniques, *e.g.*, LU decomposition and Woodbury formulas, are also adopted to speed up matrix factorization [8, 25]. This constitutes a major difference with respect to HotSpot, that adopts more complex numerical methods to achieve higher accuracy and stability. HotSpot mainly relies on the Runge-Kutta method, that goes through a number of Euler-style steps to match a Taylor series expansion, thus resulting in heavier simulation overheads [8].

Results in Table 5 also show that the Intel MKL accelerator is basically useless for block level mode simulations. The number of threads is indeed automatically determined by the MKL library, given the

Benchmark	HotSpot	HotSpot+MKL		SystemC-AMS		
	[s]	[s]	threads (#)	[s]	speedup w.r.t. Standard	speedup w.r.t. MKL
1	1.279	1.280	1	0.051	25.08x	25.09x
2	11.252	11.303	1	0.216	50.09x	52.33x
3	27.013	27.011	1	0.387	69.80x	69.80x
4	47.488	47.491	1	0.539	88.10x	88.11x
5	353.329	353.132	1	1.263	279.75x	279.60x

Table 5. Comparison of Transient Simulation Time for the block level mode.

size of the problem to be solved: since the RC networks are small (maximum 1,500 network elements), the code is considered too small to effectively partition the computation load into multiple threads. As only one thread is used, simulation times are comparable with the standard version of HotSpot.

Even if simulation speed is not our main goal, these numbers prove that the lightweight solvers of SystemC-AMS enable efficient thermal simulation in transient scenarios.

6.1.4 Accuracy. We assessed the accuracy of our simulator in terms of the relative error with respect to HotSpot, defined as: $\frac{T_{AMS}-T_{HotSpot}}{T_{HotSpot}-T_{Ambient}}$, *where* T_{AMS} *and* $T_{HotSpot}$ *are the temperatures estimated by the simulators (i.e., ambient temperature plus estimated temperature variation), and* $T_{Ambient}$ *is ambient temperature. The latter term appears at the denominator since both simulators compute thermal variation with respect to ambient temperature, rather than the actual temperature value.* The error of SystemC-AMS simulation for the steady-state case is negligible with a maximum error over all benchmarks of 10^{-6}%. This is expected since the simulation implies solving a resistive network without requiring iterations, thus avoiding the accumulation of approximations. The transient scenario is more interesting, as the thermal RC network includes also capacitors and multiple iterations are required to compute the instantaneous temperature trace over time. Table 6 shows that the average error of SystemC-AMS on all benchmarks is well below 1%, and the maximum error is about 1%.

Benchmark	Max error [%]	Avg error [%]
1	1.07	0.034
2	1.06	0.052
3	1.17	0.040
4	1.15	0.060
5	1.03	0.015

Table 6. Accuracy of transient temperature estimation for block level mode.

This proves that the SystemC-AMS solver is extremely accurate with respect to the HotSpot simulation kernel in spite of the faster execution time. This is evident also from Figures 7, that proves the high level of accuracy by showing that the temperature curves computed by SystemC-AMS and HotSpot are almost indistinguishable. For space reasons we report only the plots for Benchmark 1, but results for the other benchmarks exhibit a similar accuracy.

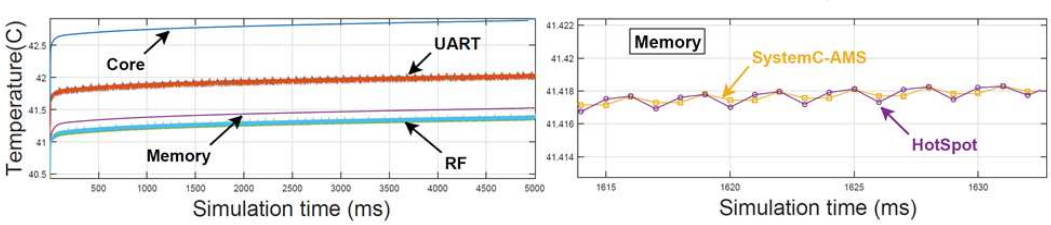


Fig. 7. Transient temperature profile computed by SystemC-AMS and HotSpot for benchmark 1.

6.2 Grid-Level Mode

When adopting grid level mode, the impact of the different benchmarks on the generated RC network becomes less evident. The size of the RC network is in fact independent from both the functional components and the floorplan, and the latter impacts only the thermal coefficients used for resistors and capacitors. For the sake of readability, in the following we will refer to Benchmark 1 for our detailed analysis; we will discuss the extension of the concepts to generic floorplans in Section 6.2.5.

6.2.1 RC Network Initialization. Table 7 reports the time necessary to setup the RC network for HotSpot and SystemC-AMS, for different grid sizes, together with the sizes of the corresponding network (number of nodes, resistors, and capacitors).

Benchmark	Nodes	Rs (#)	Cs (#)	Steady state (ms)		Transient (ms)	
				SystemC-AMS	HotSpot	SystemC-AMS	HotSpot
4x4	76	208	76	1.678	0.30	1.982	0.31
8x8	268	784	268	6.979	0.32	8.318	0.34
16x16	1,036	3,088	1,036	58.461	0.38	71.806	0.39
32x32	4,108	12,304	4,108	971.397	0.50	1,405.12	0.52
32x64	8,204	24,328	8,204	3,661.510	0.58	5,445.83	0.60
64x64	16,396	49,168	16,396	9,033.730	0.68	9,781.29	0.71

Table 7. Comparison of RC Network Initialization Time for the grid level mode.

The initialization time is very small for HotSpot, if compared to the corresponding time for block-level mode (Table 3). The reason is that the RC network construction algorithm is different for the two versions in HotSpot. Block-level mode requires to consider the layout of blocks to determine the adjacency relations between functional blocks. On the contrary, the division of the floorplan into cells of identifcal size yields a corresponding regular structure of the RC network and simplfies significantly the computation. As an example, the time to build the largest network (64x64, 16,396 nodes) requires 0.71 ms, roughly the same time required to build the block-level network of Benchmark 2, which only has 84 nodes.

The initialization time for SystemC-AMS is much larger, and grows super-linearly with the size of the network. *This is due to the fact that SystemC-AMS is a general-purpose simulation language, that thus requires to construct data structures that go beyond the sole needs of thermal simulation, e.g., including data structures related to kernel routines, modules, and equation management. At initialization time, all data structures have to be initialized and populated by querying all network elements. This constitutes a main difference with respect to HotSpot, that simply fills a matrix of coefficients, while SystemC-AMS actually* builds *the actual code of the RC network, deriving the underlying equations from the instantiated ELN primitives and their connections. Additionally, SystemC-AMS makes intensive use of dynamic memory, that allows the construction of larger networks at the price of slower access, due to dereferencing pointers. However, it is important to note that improving memory usage is not something that can be done by users, as it would require an extensive reorganization of the SystemC-AMS kernel code.*

In spite of a worst scalability of this setup phase, the time required by SystemC-AMS remains reasonable. In the case of the largest network (64x64) it takes just about 9 seconds to build the network (results in the table are in milliseconds). A 64x64 grid is considered the reference granularity for grid simulation to guarantee a good resolution in the thermal map [13], and the numbers shows that SystemC-AMS has no problem in handling this size. *Supporting larger RC networks might require an optimization of memory usage, given that the allocated heap memory grows super-linearly with grid size.*

Notice also that the initialization time is larger for the transient version, since the RC network includes also one capacitor per network node.

6.2.2 Steady-State Simulation. The overhead incurred by SystemC-AMS for the RC network construction becomes evident when analyzing the runtime of steady-state simulation. In this case, in fact, running a single iteration penalizes the SystemC-AMS implementation, as shown in Table 8.

Grid size	SystemC-AMS time (ms)	HotSpot time (ms)	HotSpot + SuperLU time (ms)
4x4	36.46	15.1	10.4
8x8	167.86	20.1	11.9
16x16	1,246.85	33.4	15.5
32x32	16,493.32	94.3	39.8
32x64	54,521.86	169.9	77.6
64x64	92,449.81	564.8	206.5

Table 8. Comparison of Steady-State Simulation Time for the grid level mode.

HotSpot is clearly more scalable and much more efficient than SystemC-AMS by about two orders of magnitude. Nevertheless, times are in all cases reasonable (values are in milliseconds): even the slowest 64x64 case for SystemC-AMS requires about 90 seconds to compute. It is worth mentioning that the combination (grid mode/steady-state simulation) is the least "interesting" of the four possible combinations (block/grid and steady-state/transient): having a single average value per grid point is not a very useful information. As a matter of fact, benchmarks for thermal simulation typically consider grid mode/transient simulation as a reference scenario [13].

Notice that HotSpot is also more efficient than in the case of steady-state simulation *in block mode*. This is due to the adoption of a more effective solver for steady-state simulation, based on the Gauss-Seidel kernel for the grid-level mode. Moreover, HotSpot steady-state simulation is additionally improved by the adoption of SuperLU, that further speeds up simulation.

6.2.3 Transient Simulation. When moving to transient simulation, the SystemC-AMS overhead for the RC network construction gets easily amortized over the large number of iterations required by the computation. Table 9 show the simulation times of SystemC-AMS and HotSpot for the case of transient simulation. The advantage of the SystemC-AMS implementation is now evident, with speedups of up to 60x. This is due to the longer simulation, which requires 6,000 simulation steps to complete. The speedup of SystemC-AMS remains roughly flat in the range 20-25% for smaller grid sizes, but grows for the largest grids. Results are consistent with those for block-level transient simulation (Table 5), although speedups are smaller in this case due to the management overhead resulting from a larger RC network. Even if simulation speedup with respect to HotSpot is not the main goal of this work, Table 9 proves that SystemC-AMS allows effective runtime evaluation of the thermal evolution of a system, thanks to the effective transient simulation.

Grid Size	HotSpot [s]	HotSpot +MKL		SystemC-AMS		
		[s]	threads (#)	[s]	Speedup w.r.t Standard	Speedup w.r.t MKL
4x4	3.508	3.510	1	0.163	21.52x	21.34x
8x8	13.438	13.092	1	0.671	20.02x	19.51x
16x16	59.903	58.899	1	3.019	19.84x	19.12x
32x32	779.093	778.285	3	30.735	25.31x	25.00x
32x64	4,966.469	4,866.789	5	145.464	34.14x	33.46x
64x64	11,849.712	11,827.830	9	191.176	61.98x	61.78x

Table 9. Comparison of Transient Simulation Time for the grid level mode.

Notice that in this case HotSpot does not benefit from architecture-specific libraries, such as the MKL. *The versions with grid size up to 16x16 are still considered too small to benefit from parallelization*

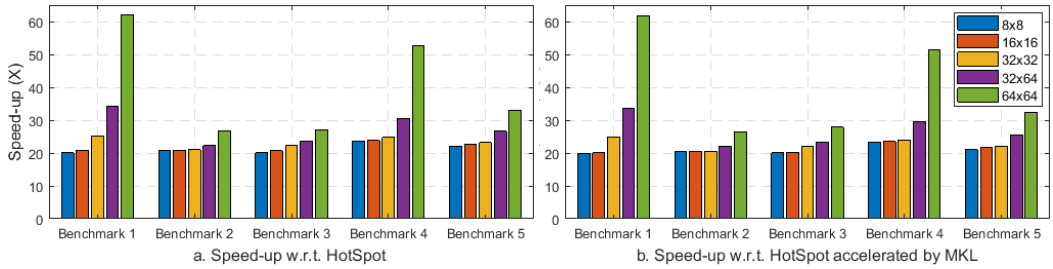


Fig. 8. Speed-up of SystemC-AMS code with respect to HotSpot (a) and its MKL accelerated version (b).

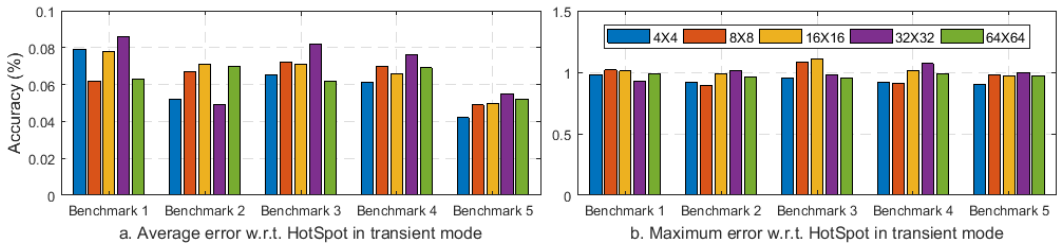


Fig. 9. Average (a) and maximum (b) error of SystemC-AMS with respect to HotSpot for transient simulation.

by the MKL, and thus they run on a single thread. The other versions are run on multiple threads (up to 9), given the larger number of RC network elements. However, the high synchronization overhead of the pthread library basically nullifies the potential of the parallel code [5], yielding simulation times only slightly lower than the sequential version of HotSpot.

As mentioned in Section 6.2, the above analysis uses Benchmark 1 as the underlying circuit. In order to see if the considerations apply also to other benchmarks, we also ran the other four benchmarks with the five grid sizes. Results are summarized in Figure 8, that plots the speed-up of SystemC-AMS over HotSpot and its MKL-accelerated version for the five benchmarks. The plots clearly show the trends of a speedup that increases with the number of grid cells applies to all benchmarks. However, the range of the speed-ups is quite circuit-dependent; the largest one is for Benchmark 1 (of about 60x), while the lowest ones are for Benchmarks 2 and 3 (about 25x). This difference in speedup will be elaborated in Section 6.2.5.

6.2.4 Accuracy. Table 10 reports the accuracy of SystemC-AMS in grid-level mode compared with respect to HotSpot, again using the results of Benchmark 1 as a reference, and shows the average and the maximum error of each configuration. *Steady state simulation* proved to be extremely accurate also in grid level mode, with errors lower than 0.00001%. *Transient simulation* exhibits a larger error, due to the presence of multiple simulation steps. However, the average error is still well below 1%, and the maximum error is only 1%.

Grid size	Steady State Temperature		Transient Temperature	
	Avg.error (%)	Max.error (%)	Avg.error (%)	Max.error (%)
4x4	0.23e-5	0.66e-5	0.079	0.980
8x8	0.15e-5	0.56e-5	0.062	1.020
16x16	0.19e-5	0.79e-5	0.078	1.010
32x32	0.16e-5	0.68e-5	0.086	0.930
32x64	0.21e-5	0.72e-5	0.055	1.050
64x64	0.13e-5	0.51e-5	0.063	0.990

Table 10. Accuracy of Steady state and transient temperature of test bench 1 for the grid level mode.

6.2.5 Dependence on Floorplan, Power Variance and Simulation Time Step. In this section we analyze the sensitivity of the thermal simulation with respect to (i) the floorplan, (ii) the the distribution of the power values, and (iii) the adopted simulation time step. It is intuitive that these factors might affect the performance of the simulator because the former determines the values of the electrical elements, which can impact simulation time, whereas the latter two might affect the number of iterations required to reach convergence. To this purpose, we ran three sets of experiments to illustrate how these factors affect the simulation speed. Due to space constraints, in this sections we will apply the analysis only to Benchmark 1. However, the same considerations hold for all benchmarks.

Impact of floorplan. The main impact of varying the floorplan is that the values of resistances and capacitances change, even if the reminder of benchmark configuration is unchanged. Indeed, such values strictly depend on the area of each grid cell (in grid mode) and on the thickness of each layer in the model. We thus decided to explore the behavior of the simulators by artificially modifying the floorplans, by inserting blank spaces. In other words, this experiment could be seen as an analysis of the sensitivity of the simulator to the *utilization of the floorplan*, *i.e.*, whether a denser or sparser floorplan affects simulation time.

Grid size	Floorplan 1	Floorplan 2	Floorplan 3
Width [mm]	3.80	6.84	9.45
Height [mm]	5.63	7.71	8.33
Utilization [%]	93.69	78.82	62.73

Table 11. Width and height of different floorplans.

Table 11 shows the width and height of three different floorplans used and the resulting utilization. The different floorplan configuration and the presence of blank spaces affects the computed RC network, as the values of resistors and capacitors are different across the floorplans. Table 12 reports the corresponding simulation time on the three floorplans, when varying grid granularity. The table highlights that SystemC-AMS is consistent across all floorplans, as simulation time is mainly dominated by the size of the RC network. On the contrary, HotSpot simulation time varies across the benchmarks (this is more evident for larger RC networks).

Grid size	HotSpot			SystemC-AMS		
	Floorplan 1	Floorplan 2	Floorplan 3	Floorplan 1	Floorplan 2	Floorplan 3
4x4	3.521	3.491	3.517	0.163	0.162	0.163
8x8	13.573	13.599	13.521	0.671	0.671	0.672
16x16	59.876	59.916	59.862	3.019	3.016	3.015
32x32	779.703	543.779	287.742	30.735	30.733	30.736
32x64	4,967.191	2,506.741	1,788.125	145.464	145.460	145.462
64x64	11,851.297	5,486.625	3,639.255	191.176	191.171	191.177

Table 12. Simulation of HotSpot and SystemC-AMS for benchmark 1 for the floorplans in Table 11.

To explain this behavior, it is necessary to look deeply into the solvers. SystemC-AMS uses simple solvers, like Euler and trapezoidal methods, that have a fixed number of iterations at any time step. On the other hand, HotSpot uses the 4th order Runge Kutta method, that performs a number of iterations that varies with the slope of temperature variation. At any time instant, in HotSpot the slope for a given cell is directly proportional to power consumption P of the cell and to the gradient of the temperature with respect to the adjacent cells i, and inversely proportional to the values of resistances and of cell capacitance:

$$\frac{dT}{dt} = \left(P - \sum_{\mathrm{i}} \frac{T - T_i}{R_i} \right) \cdot \frac{1}{C} \tag{2}$$

Grid size	Constant (s)	10% Variance (s)	50% Variance (s)
4x4	0.162	0.163	0.162
8x8	0.670	0.672	0.672
16x16	3.016	3.016	3.018
32x32	30.733	30.734	30.734
32x64	145.460	145.460	145.461
64x64	191.177	191.177	191.176

Table 13. SystemC-AMS simulation time for different power traces.

An effect of this dependence is that larger values of Rs and Cs make convergence faster, and thus require fewer solver iterations. At any time step, HotSpot conducts multiple steps of the Runge Kutta algorithm, and the number of iterations is adaptive: when the slope is steep, it is possible to decrease the number of iterations and to accordingly decrease simulation time significantly. Vice versa, less steep slopes require more iterations to converge. This is proven by the fact that the denser floorplans (*e.g.*, Floorplan 1) result into longer simulation times: the smaller values for Rs and Cs slow down convergence. Conversely, the simulation time is up to 3 times faster for the less utilized floorplan (Floorplan 3). Since optimized floorplans are denser for obvious area and performance reasons, the speedup figures shown so far are conservative. Note that the floorplan density of the benchmarks adopted throughout the experimental analysis is similar to the one of Floorplan 1 (average density 96.64%, see Table 2).

To further argument this result, we deeply analysed the HotSpot execution flow, to extract the number of iterations of the Runge Kutta solver over the whole simulation. To get significative numbers, we focused on the 64x64 grid granularity, whose simulation times vary significantly over the three floorplans. Floorplan 1 takes 11,851.715s, as a result of 969,568 Runge Kutta iterations. Simulation is 2.16x faster for Floorplan 2, that requires 571,352 Runge Kutta iterations. Finally, Floorplan 3 requires only 462,582 Runge Kutta iterations, thus resulting in a speedup of 3.26x with respect to the initial floorplan. These numbers highlight the impact of the floorplan over HotSpot simulation, and the weight of the Runge Kutta iterations over the resulting simulation time.

Impact of power traces. To prove that SystemC-AMS performance is determined only by the size of the RC network, we also investigate the impact of the input power traces on simulation time. We artificially created synthetic traces for the components of benchmark 1. All traces have the same average value (equal to the average value of the original power trace). The three traces are then created by varying the variance of the power values: constant power trace, 10% variance and 50% variance.

Table 13 reports the corresponding SystemC-AMS simulation time. The table proves that the times are consistent, and that they are not affected by the different distribution of power values. This confirms that the SystemC-AMS performance is sound: it is affected only by the size of the RC network, and thus by the number of network elements. Other dimensions, like floorplan density or input power distribution, do not impact on simulation time, thus ensuring consistent performance and that no worst-case simulation scenarios do exist.

Impact of simulation time step. The last analysis focuses on the impact of the simulation time step, i.e., the step at which power values are fed into the simulator, and at which the corresponding temperature values are expected in output. Table 14 reports the simulation time of HotSpot and SystemC-AMS when running 6,000 simulation steps of variable length for the 64x64 configuration.

The table highlights that the behavior of SystemC-AMS and HotSpot are different for a given value of time step. SystemC-AMS is basically insensitive to the step value across all simulations. As a matter of fact, SystemC-AMS recomputes the RC network once per time step, and the number of time steps is the same across all simulations (i.e., 6,000). On the contrary, HotSpot simulation

Time step	Simulation		HotSpot		SystemC-AMS		
	Steps (#)	Length (ms)	Time (s)	Runge Kutta (#)	Time (s)	Speedup	*Avg. error (%)*
1 ms	6,000	6,000	11,846.87	969,572	191.17	61.97x	0.097
100 us	6,000	600	1,864.71	116,112	191.72	9.73x	0.018
10 us	6,000	60	277.95	29,866	191.92	1.45x	0.015
1 us	6,000	6	149.13	24,224	191.67	0.78x	0.014

Table 14. Simulation time for SystemC-AMS and HotSpot for different time step sizes.

time varies significantly across the experiments, with simulation times from about 60x slower to values slightly smaller than those of SystemC-AMS. This can be explained by recalling that HotSpot performs a number of iterations of the Runge Kutta solver for each time step, until convergence is reached. The time step used by HotSpot for such iterations is adaptive, with a minimum value set by default to 100ns: as a consequence, the larger the time step, the more Runge Kutta iterations will be necessary to achieve convergence. To prove this, we extracted the number of Runge Kutta iterations: given the same number of simulated time steps, HotSpot spends almost one billion iterations for time step 1ms, and less than 25,000 for time step 1us.

As a consequence, the performance of SystemC-AMS over HotSpot strictly depends on the simulation time step. SystemC-AMS has runtimes comparable with HotSpot on cycle accurate simulations (e.g., 1us-10us), typical of architectural or RTL simulations. However, for system-level simulations in which we need to evaluate transient behavior over longer time intervals (in the milliseconds to second range), SystemC-AMS sensibly outperforms HotSpot, with speedups of up to two orders of magnitude.

The table highlights that the time step impacts also on the accuracy of the SystemC-AMS simulation w.r.t. HotSpot. This can be explained by considering that the larger the time step, the more Runge Kutta iterations will be necessary to achieve convergence (Table 14). Indeed, this implies that the result achieved by the SystemC-AMS solver will be less accurate with the larger time steps, as each iteration of Runge Kutta is almost equivalent to one Euler-style step. Vice versa, when the time step is smaller, the Runge Kutta method used by the HotSpot solver will require fewer iterations to converge, thus behaving similarly to the SystemC-AMS solver and reducing the sources of errors.

6.3 Comparison of Block-Level Mode and Grid-Level Mode

Even if the underlying RC network model is the same, the block mode and the grid mode provide two different granularities to thermal analysis, with effects on the characteristics of thermal simulation.

By comparing the accuracy of the two configurations (Tables 6 and 10), it is evident that the the error is almost independent from the granularity. Accuracy is indeed affected whenever the HotSpot solver behaves differently from the SystemC-AMS solver (e.g., requires more Runge Kutta iterations), so that the solvers converge on different results. Given that granularity does not impact on the behavior of the solvers, the block mode behaves similarly to the grid mode, and the error is thus mostly determined by the simulation mode: average error is 1.5 10^{-6}% for steady-state simulation and 0.05% for transient, and maximum error is 6 10^{-6}% and about 1%, respectively.

When looking at the speed-up, it is easy to notice that the performance of both block mode and grid mode is always determined by the number of RC network elements, but there is no evident correlation between the two simulation modes. The difference is due to HotSpot, that has two different algorithms to build the RC network model: the block mode version requires to take into account at any time the adjacencies between functional components and their irregular shape, thus affecting simulation performance. Vice versa, computation is heavily simplified for the grid mode, where all grid cells have the same size and their adjacency relations are pre-defined. This improves the performance of HotSpot in the grid mode, and thus lowers the speed-up of SystemC-AMS.

6.4 IEEE PATMOS 2017 Thermal Simulation Contest

This section applies the proposed SystemC-AMS thermal simulators to the benchmarks proposed by the 2017 IEEE PATMOS contest [13]. Table 15 reports the characteristics of all benchmarks. It is important to note that this experiment is especially useful, as it allows to evaluate the characteristics of the thermal simulators on a typical configuration and on non-trivial benchmarks.

Given the sensitivity of HotSpot with respect to floorplan, we considered two different floorplans for the benchmarks: an area-optimized floorplan (generated with the HotFloorplan tool, column *Optimized floorplan*), and a thermal-aware floorplan (provided by the contest, column *Thermal-aware floorplan*). The main difference between the floorplans is area utilization, which is on average 91,90% for the optimized version and 75.15% for the thermal-aware version, respectively.

Benchmark	Blocks	Thermal-aware Floorplan		Optimized Floorplan	
	[#]	Width × Height [mm]	Utilization [%]	Width × Height [mm]	Utilization [%]
PATMOS 1	30	8.00 × 6.64	66.5	7.85 × 4.75	94.8
PATMOS 2	30	10.56 × 6.05	66.2	8.34 × 5.26	96.3
PATMOS 3	20	4.82 × 6.11	83.7	5.16 × 5.07	94.1
PATMOS 4	60	9.46 × 9.64	73.7	8.90 × 9.19	82.2
PATMOS 5	25	5.56 × 5.85	77.0	5.23 × 4.98	96.1
PATMOS 6	35	7.91 × 5.15	86.7	6.27 × 6.05	93.2
PATMOS 7	40	7.94 × 7.72	78.4	8.05 × 6.22	90.0
PATMOS 8	40	8.38 × 6.46	80.9	8.09 × 5.95	91.0
PATMOS 9	45	8.28 × 8.23	71.3	7.16 × 7.64	90.8
PATMOS 10	50	8.34 × 8.15	66.8	6.83 × 7.37	90.2

Table 15. Characteristics of the RC thermal networks of the IEEE PATMOS contest benchmarks.

The scenario of interest defined by the contest is a *6000 ms transient simulation for a 64x64 grid*. To further support the claim of flexibility of our method, we ran SystemC-AMS and HotSpot (with and without MKL-accelerator) on these benchmarks with a time step of 1ms. RC network initialization performance is aligned with the numbers in Table 7, with an average time of about 9 seconds for SystemC-AMS and of 2 ms for both versions of HotSpot.

Tables 16 and 17 report the simulation performance of transient simulation on the thermal-aware and the area-optimized floorplans, respectively. From the tables it is evident that *SystemC-AMS performance* is consistent with the one reported for the 64x64 granularity in Table 9, about 190 seconds for all benchmarks[2]. As a matter of fact, for all SystemC-AMS simulations, the difference over all benchmarks for both floorplans is less than 0.5%. This independence of the benchmark (in particular, its power distribution) and of the floorplan is an important property of the SystemC-AMS solver. Performance only depends on the size and the connections of the RC network.

The same does not hold for HotSpot, as already demonstrated in the previous section. Simulation times, besides being again much longer than those of our method (28x to 35x, depending on the floorplan density), vary sensibly across different benchmarks (about 5300s difference between the slowest and fastest case, more than a 2x variation). Benchmarks with less steep temperature curves have faster convergence and less iterations of the solver functions (*e.g.*, benchmark 4). Viceversa, simulation times grow when convergence requires more solver steps (*e.g.*, benchmark 3).

6.5 Concurrent Simulation of Functionality, Power, and Temperature

An additional strength of the proposed approach is the possibility to run thermal simulation in parallel with respect to functional and power simulation. As a final experiment, we thus implemented

[2]Note that the simulations have the same length

Benchmark	SystemC-AMS time (s)	HotSpot		HotSpot + MKL		
		time (s)	speedup (x)	threads (#)	time (s)	speedup (x)
PATMOS 1	192.174	5,262.99	27.39	9	5,166.33	26.88
PATMOS 2	191.793	5,089.78	26.53	9	5,031.42	26.23
PATMOS 3	191.374	9,456.60	49.41	9	9,357.12	48.89
PATMOS 4	194.681	3,082.81	15.84	9	3,002.41	15.42
PATMOS 5	192.831	8,127.52	42.15	9	8,084.80	41.93
PATMOS 6	191.888	6,961.85	36.28	9	6,901.22	35.96
PATMOS 7	192.617	4,148.77	21.54	9	4,078.94	21.18
PATMOS 8	192.859	5,296.57	27.46	9	5,216.96	27.05
PATMOS 9	193.514	4,172.22	21.56	9	4,072.62	21.05
PATMOS 10	194.553	4,181.85	21.49	9	4,095.31	21.50
Average			28.96	-		28.61

Table 16. Transient Simulation Time for the benchmarks adopted by the IEEE PATMOS design contest with the thermal-aware floorplans.

Benchmark	SystemC-AMS time (s)	HotSpot		HotSpot + MKL		
		time (s)	speedup (x)	threads (#)	time (s)	speedup (x)
PATMOS 1	193.261	7,979.98	41.29	9	7,868.74	40.64
PATMOS 2	192.188	6,629.55	34.50	9	6,558.03	34.12
PATMOS 3	192.287	10,574.82	54.99	9	10,415.24	54.16
PATMOS 4	193.846	4,367.72	22.53	9	4,289.86	22.13
PATMOS 5	192.605	10,508.38	54.56	9	10,463.62	54.33
PATMOS 6	191.918	7,374.79	38.43	9	7,227.53	37.66
PATMOS 7	192.622	5,327.02	27.65	9	5,271.11	27.37
PATMOS 8	192.994	5,972.41	30.95	9	5,873.77	30.43
PATMOS 9	193.226	5,001.62	25.88	9	4,983.08	25.79
PATMOS 10	193.892	5,158.78	26.61	9	5,040.02	25.99
Average			35.74	-		35.26

Table 17. Transient Simulation Time for the benchmarks adopted by the IEEE PATMOS design contest with optimized floorplans.

a scenario in which the SystemC-AMS thermal simulation of case study 1 (Figure 4.a) is run concurrently with functional and power models, by adopting the framework proposed in [32] (left-hand side of Figure 10). To this extent, we implemented all functional components of case study 1 as SystemC modules, whose functional evolution drives a model of power consumption. In detail, the core implements an instruction-based power model similar to [29] (load/store instructions require 2 mW, arithmetic instructions 1 mW, branches 0.4 mW and NOPs 0.1 mW). The power consumption of the other components depends on the device state, *e.g.*, the UART consumes 9.98 μW when idle and 1.43 mW when transmitting or receiving.

The right-hand side of Figure 10 shows an excerpt of the simulation, by focusing on the current consumption (top) and thermal profile (middle) of the core, and on the thermal profile of the UART (bottom) over 150ms. During this interval, the UART is in idle mode to avoid cross interference. It is worth re-emphasizing that curves are obtained as traces of a *single simulation run*; SystemC automatically manages the appropriate model of computation for different description styles.

The solid blue lines highlight the dependency between power consumption and temperature. A time slot of high activity of the core determines a larger current demand for a prolonged time. The corresponding increase of temperature of the core (arrow 1) affects also the neighbours, as highlighted by the small fluctuations of the temperature of the UART (arrow 3).

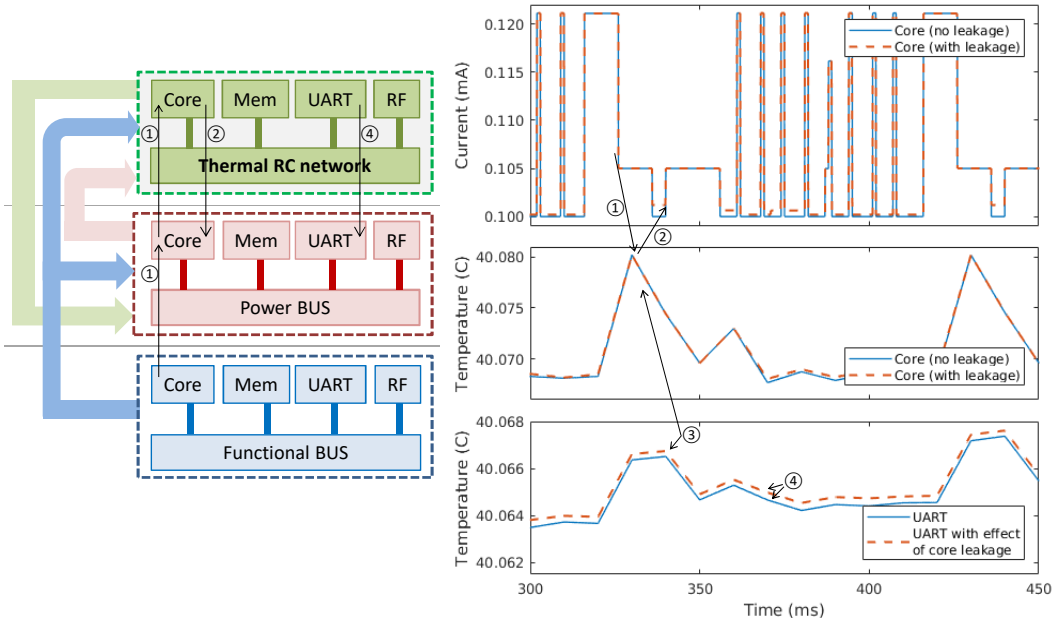


Fig. 10. Power and thermal profile of the core and the UART of case study 1. The arrows highlight the mutual dependence between power consumption and thermal evolution.

To show also the impact of temperature over power, we extended the power model of the core with the dependence of static power from temperature (dashed orange lines). In this scenario, arrow (2) highlights that higher component temperature is reflected as a higher static power consumption during idle periods. As an additional side effect, arrow (4) highlights that modifications in the power model of one component affect the thermal profile also of neighbours, as the higher static power of the core slightly increases the temperature of the UART. This experiment proves that the adoption of SystemC-AMS for the thermal simulator allows to reproduce mutual dependencies between power and temperature, and to get a more accurate picture of system evolution. Such a run-time tracking could not be simulated with trace-based approaches, which usually estimate temperature from power traces, thus loosing the cyclic dependence.

As a further analysis, we investigated the possibility of replacing the SystemC-AMS RC network with HotSpot. This was achieved by compiling the HotSpot library to generate a static library, and by replacing the SystemC-AMS RC network with invocations to the HotSpot library functions. All the rest of the system, including the dependencies between power and temperature, have not been modified, so to evaluate only the impact of the thermal simulator.

We run both simulations for 6,000ms which is same length as all previous simulations. The version including HotSpot took 4,165s, while the system implemented entirely in SystemC-AMS took only 673s, with a speed up of 6.2×. Note that the speedup is lower than for the transient simulation shown in Section 6.1.3 (25×), as an effect of the additional simulation overhead due to the functional and the power layers. However, adopting SystemC-AMS for thermal simulation still allows good simulation performance, together with an easier integration flow.

To confirm this, we run a longer simulation (for 600,000ms). Here the speedup increases up to 57.4×: SystemC-AMS takes 6,508s, and the version including HotSpot takes 373,417s. This implies that the SystemC-AMS-based solution scales better on longer simulations, a 100x longer simulation time required only 10x longer with our approach. These experiments thus confirm the effectiveness

of SystemC-AMS for the concurrent simulation of multiple aspects of a system: a good accuracy and an easy integration process is indeed accompanied by good simulation performance.

6.6 Summary and Discussion

Our analysis highlighted that SystemC-AMS has several strengths over state of the art of thermal simulation:

- SystemC-AMS, as a general purpose simulator, allows simulating the thermal evolution concurrently with functional and power simulation, as well to reliability, with the advantage of straightforward integration and support for the modeling of mutual interactions;
- Effective simulation performance:
  - The only weakness of SystemC-AMS for thermal simulation is its higher overhead at initialization time when the thermal network gets large. However, this is an issue only when considering steady-state simulation, which is a less interesting scenario from the point of view of thermal analysis, since it yields a single number per element (grid or block) determined by the average power of that element;
  - Steady-state simulation is still feasible even for the largest grid sizes;
  - For transient thermal simulation, SystemC-AMS yields speedups between 15x to 60x for grid-level simulation and up to 280x for block-level simulation;
  - SystemC-AMS simulation time is virtually insensitive to the power distribution of the underlying circuit and of the floorplan and only depends on the number of blocks (block-level mode) or the number of grid points (grid-level mode), that vice versa heavily affect HotSpot;
- Despite of relying on simpler solvers, SystemC-AMS simulations yield remarkable accuracy, with maximum error over all possible scenarios in the order of 1%.

7 CONCLUSIONS

This work proposed the adoption of SystemC-AMS for the runtime monitoring of temperature, with the goal of improving extra-functional property evaluation at all design stages. The generated code exploits the ELN abstraction level, and it does not require the development of ad-hoc circuit simulators. The experimental analysis proved the effectiveness of the generated code. The initialization overhead, due to the general-purpose nature of SystemC-AMS, is indeed compensated by speedups up to two orders of magnitude in case of transient simulation. This, together with a high level of accuracy (max. error in the order of 1%), allows to effectively evaluate the thermal evolution, also in case of complex benchmarks. Additionally, the SystemC-AMS thermal simulator can be simulated in a single run with power and functional models, thus allowing to reproduce mutual dependencies among different aspects of a smart system. Future work will focus on additional refinements to the methodology, to reduce the initialization and memory overhead.

REFERENCES

[1] B. Beckmann, Y. Eckert, et al. 2013. A Comprehensive Timing, Power, Thermal, and Reliability Model for Exascale Node Architectures. In *Proc. of DOE MODSIM*. 1–3. cseweb.ucsd.edu/~marora

[2] L. Benini, R. Hodgson, and P. Siegel. 1998. System-level power estimation and optimization. In *Proc. of ACM/IEEE ISLPED*. 173–178.

[3] T. Bouhadiba, M. Moy, F. Maraninchi, J. Cornet, L. Maillet-Contoz, and I. Materic. 2013. Co-simulation of Functional SystemC TLM Models with Power/Thermal Solvers. In *Proc. of IEEE IPDPSW*. 2176–2181.

[4] L. Bousquet and E. Simeu. 2013. System-level modeling of electromechanical devices with energy consumption. In *Proc. of IEEE SysCon*. 756–761.

[5] D. Buttlar, J. Farrell, and B. Nichols. 2013. *PThreads Programming - A POSIX Standard for Better Multiprocessing.* O'Reilly Media.

[6] Y. Chen, S. Vinco, E. Macii, and M. Poncino. 2016. Fast thermal simulation using SystemC-AMS. In *Proc. of ACM/IEEE GLSVLSI.* 427–432.

[7] F. Fummi, M. Lora, F. Stefanni, D. Trachanis, J. Vanhese, and S. Vinco. 2014. Moving from co-simulation to simulation for effective smart systems design. In *IEEE/ACM DATE.* 1–4.

[8] P. A. Hartmann, P. Reinkemeier, A. Rettberg, and W. Nebel. 2009. Modelling control systems in SystemC AMS - Benefits and limitations. In *Proc. of IEEE SOCC.* 263–266.

[9] N. Hatami, R. Baranowski, et al. 2014. Multilevel Simulation of Nonfunctional Properties by Piecewise Evaluation. *ACM TODAES* 19, 4 (2014), 37:1–37:21.

[10] M. Hsieh, K. Pedretti, J. Meng, et al. 2012. SST + Gem5 = a Scalable Simulation Infrastructure for High Performance Computing. In *Proc. of ACM SIMUTOOLS.* 196–201.

[11] W. Huang, K. Sankaranarayanan, R. J. Rib, M.R. Stan, and K. Skadron. 2007. An Improved Block-Based Thermal Model in HotSpot 4.0 with Granularity Considerations. (2007). citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.80.3864.

[12] IEEE. 2016. IEEE Standard for Standard SystemC Analog/Mixed-Signal Extensions Language Reference Manual. In *Std 1666.1-2016.* 1–236.

[13] IEEE PATMOS. 2017. DESIGN CONTEST – Design for Performance and Thermal Efficiency. (2017). http://patmos2017.web.auth.gr/patmos2017_designContest.php.

[14] Intel. 2017. MKL – Math Kernel Library (2017.2.174). (2017). https://software.intel.com/en-us/intel-mkl.

[15] L. Jani and A. Poppe. 2017. Adaptive co-simulation of functional-thermal behaviour of integrated circuits. In *IEEE THERMINIC.* 1–8.

[16] T. Kemper, Y. Zhang, Z. Bian, and A. Shakouri. 2006. Ultrafast Temperature Profile Calculation in IC Chips. In *IEEE THERMINIC.* 133–137.

[17] S. S. Kumar, A. Zjajo, and R. v. Leuken. 2015. Ctherm: An Integrated Framework for Thermal-Functional Co-simulation of Systems-on-Chip. In *Proc. of Euromicro PDP.* 674–681.

[18] X. S. Li. 2005. An Overview of SuperLU: Algorithms, Implementation, and User Interface. *ACM TOMS* 31, 3 (2005), 302–325.

[19] P. Liu, Z. Qi, H. Li, L. Jin, W. Wu, S.X. Tan, and J. Yang. 2005. Fast thermal simulation for architecture level dynamic thermal management. In *IEEE ICCAD.* 639–644.

[20] W. Liu, A. Calimera, A. Macii, et al. 2013. Layout-Driven Post-Placement Techniques for Temperature Reduction and Thermal Gradient Minimization. *IEEE TCAD* 32, 3 (2013), 406–418.

[21] J. Nayfach and J. Renau. 2009. SOI, Interconnect, Package, and Mainboard Thermal Characterization. *IEEE ISLPED*, 327–330.

[22] M. Ozisik. 1968. *Boundary Value Problems of Heat Conduction.* Oxford University Press.

[23] X. Pan, J.M. Molina, and C. Grimm. 2015. Modeling power consumption at system-level for design of power integrity-aware AMS-circuits. In *Proc. of ECSI/IEEE FDL.* 1–8.

[24] M. Pedram and S. Nazarian. 2006. Thermal Modeling, Analysis, and Management in VLSI Circuits: Principles and Methods. *Proc. of the IEEE* 94, 8 (2006), 1487–1501.

[25] C. Reuther and K. Einwich. 2012. A SystemC AMS extension for controlled modules and dynamic step sizes. In *Proc. of IEEE/ECSI FDL.* 90–97.

[26] K. Skadron, M.R. Stan, Wei Huang, et al. 2003. Temperature-aware computer systems: Opportunities and challenges. *IEEE Micro* 23, 6 (2003), 52–61.

[27] K. Skadron, M. R. Stan, B. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. 2003. *Temperature-Aware Microarchitecture: Extended Discussion and Results.* Technical Report CS-2003-08. Univ. of Virginia Dept. of CS.

[28] L. Thiele, L. Schor, H. Yang, and I. Bacivarov. 2011. Thermal-aware system analysis and software synthesis for embedded multi-processors. In *ACM/EDAC/IEEE DAC.* 268–273.

[29] V. Tiwari, S. Malik, A. Wolfe, and M.T.C. Lee. 1996. Instruction level power analysis and optimization of software. *IEEE ICVD* 13, 2 (1996), 223–238.

[30] A. Viehl, B. Sander, O. Bringmann, and W. Rosenstiel. 2008. Integrated requirement evaluation of non-functional system-on-chip properties. In *ECSI/IEEE FDL.* 105–110.

[31] A. Vincenzi, A. Sridhar, M. Ruggiero, and D. Atienza. 2011. Fast thermal simulation of 2D/3D integrated circuits exploiting neural networks and GPUs. In *IEEE/ACM ISLPED.* 151–156.

[32] S. Vinco, Y. Chen, F. Fummi, E. Macii, and M. Poncino. 2017. A Layered Methodology for the Simulation of Extra-Functional Properties in Smart Systems. *IEEE TCAD* 36, 10 (2017), 1702–1715.

[33] S. Vinco, A. Sassone, et al. 2014. An Open-source Framework for Formal Specification and Simulation of Electrical Energy Systems. In *Proc. of IEEE/ACM ISLPED.* 287–290.

[34] H. Wang, J. Ma, S. Tan, C. Zhang, H. Tang, K. Huang, and Z. Zhang. 2016. Hierarchical Dynamic Thermal Management Method for High-Performance Many-Core Microprocessors. *ACM TODAES* 22, 1 (2016), 1:1–1:21.

[35] H. Wang, J. Wan, S. X. D. Tan, C. Zhang, H. Tang, Y. Yuan, K. Huang, and Z. Zhang. 2018. A Fast Leakage-Aware Full-Chip Transient Thermal Estimation Method. *IEEE TC* 67, 5 (2018), 617–630.

[36] T.-Y. Wang and C.C.P. Chen. 2004. SPICE-compatible thermal simulation with lumped circuit modeling for thermal reliability analysis based on modeling order reduction. In *IEEE ISQED*. 357–362.

[37] Z. Wang, S. Kanwal, L. Wang, and A. Chattopadhyay. 2017. Automated High-level Modeling of Power, Temperature and Timing Variation for Microprocessor. *KMUTNB IJAST* 10, 3 (2017), 163–175.

[38] Y. Yang, Z. Gu, C. Zhu, R.P. Dick, and L. Shang. 2007. ISAC: Integrated Space-and-Time-Adaptive Chip-Package Thermal Analysis. *IEEE TCAD* 26, 1 (2007), 86–99.

[39] Y. Zhan, S.V. Kumar, and S.S. Sapatnekar. 2008. Thermally Aware Design. *FTEDA* 2, 3 (2008), 255–370.

[40] R. Zhang, M. R. Stan, and K. Skadron. 2015. *HotSpot 6.0: Validation, Acceleration and Extension.* Technical Report CS-2015-04. Univ. of Virginia Dept. of CS.