Modelling and Simulation Challenges and Solutions in Cooling Systems for Nanoscale Integrated Circuits

Federico Terraneo, *Member, IEEE,* Alberto Leva, *Member, IEEE,* William Fornaciari, *Senior Member, IEEE,* and David Atienza, *Fellow, IEEE*

***Abstract*—The power density in modern Integrated Circuits (ICs) is tremendous. For example, Multi-Processor Systems-on-Chip (MPSoCs) nowadays undergo temperature swings of 40 degrees in 100 milliseconds or less, with rapidly emerging and vanishing sub-millimeter hot spots. As such, not only a simulation-based cooling assessment is vital, but one has to simulate the on-chip thermal phenomena *jointly* with the heat dissipation system — historically, a challenge. In recent years, however, the idea of coupling traditional 3D chip simulators with heat dissipation models written in Equation-Based Modelling (EBM) languages has proven to be a game changer. EBM languages allow one to compose a model by assembling components described in terms of Differential and Algebraic Equations (DAE) and have the simulation code generated automatically. In this paper, we take a tutorial viewpoint on the matter just sketched, to put the reader in the position of exploiting the above technology. We also present the first *nucleus* of a model library for cooling systems, that we release as free software for the scientific and engineering community.**

I. Introduction

THE unprecedented power density of modern ICs, such as MPSoCs, has created the need for a new generation of heat dissipation systems. These include a plethora of technologies, such as liquid and two-phase cooling, Peltier elements, evaporative systems, and more. Also, the elements just mentioned are often combined, composing multi-physics cooling solutions. The relevance of the entailed problems can be appreciated by looking for example at recent works in the MPSoC domain, such as [1]–[4].

Since an incorrect behaviour of the cooling mechanism can nowadays have a profound impact on the performance and reliability of a computing system, simulation-based heat dissipation assessments are mandatory. Given the fast dynamics of the involved thermal phenomena [5], such assessments require to simulate the said phenomena together with the cooling system, as well as with the involved thermal/power/performance policies aboard the chip, see e.g. [6], [7]. Because of this new *scenario*, modelling and simulation of heat dissipation systems need a qualitative leap from several viewpoints, most notably computation speed, accuracy, and model maintainability.

The order to fulfil is tall. The problems to address are inherently cyber-physical, owing to the presence of hydraulics and thermodynamics [8] (physical) jointly with cooling circuit control algorithms and on-chip policies (cyber). Moreover, the physical part is always multi-domain, and shows so wide a variety of configurations to make a component-based approach mandatory. Also, the dimension of the systems to simulate can be large, as fine-grained spatial resolutions may be in order. At the same time, finally, the level of modelling detail must be scalable, to achieve the maximum computational efficiency in each simulation study.

Indeed, traditional thermal simulation approaches are unsuitable for such new IC operating *scenarii*. These approaches are based on exploiting the peculiarities of dynamic thermal modelling when applied to the case of ICs. As a result, they do achieve fast simulation, but at the deliberate expense of modelling generality: the faster a simulator conceived this way runs, the narrower the set of cases it can represent is. Modern alternatives such as Equation-Based Modelling (EBM) suffer from the symmetric problem: they naturally lend themselves to representing the heterogeneous physics encountered when characterising the transient thermal behaviour of cooling systems but pay for this capability in terms of computational efficiency.

Recently, we tried to join the best of the two approaches just mentioned. We made the well-assessed 3D-ICE thermal simulator capable of performing co-simulation with object-oriented, equation-based modelling and simulation tools [9]. As such, IC designers can now build simulators in which 3D-ICE takes care of the chip thermal model and of interfacing with thermal policies, while the cooling system model is assembled on a per-component basis, where components can be described in an equation-based manner. We coupled the above modelling approaches in 3D-ICE 3.0 through the Modelica language [10], [11] and the Functional Mock-up Interface (FMI) standard [12], [13]. However, the underlying ideas are general with respect to the used tools.

Coming to the goal of this paper, the evidenced revolution in the IC – thus MPSoC – cooling *scenario* also has a particularly important cultural consequence. In the past, modelling knowledge was practically needed only on the part of simulation tool developers; for example, to use 3D-ICE, an MPSoC designer just had to compile configuration files about the

F. Terraneo, A. Leva and W. Fornaciari are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy; e-mail {federico.terraneo,alberto.leva,william.fornaciari}@polimi.it.

D. Atienza is with the Embedded Systems Laboratory, École Polythechnique Fédérale de Lausanne, Station 11, CH-1015 Lausanne, Switzerland; e-mail david.atienza@epfl.ch.

Preprint version, link to article https://doi.org/10.1109/MCAS.2023.3234727

chip design, which requires no knowledge about heat model equations and their solution procedure. Now, this is not fully applicable anymore. Not only has the cooling physics become very heterogeneous, but also individual cooling systems are different from one another. This makes it impossible to provide a "standard" model to tailor with configuration files and would require the analyst to set up a completely new simulator, solution procedure included, for every new case.

As doing so would be practically infeasible, the natural way to go is not to provide standard "monolithic" cooling simulators to configure, but instead to offer *libraries* of their components (such as pumps, valves, Peltier elements and so on) for the analysts to assemble their own cooling system model for the case at hand. Quite intuitively – as will be shown here as well – assembling a model this way cannot be done with zero knowledge of the involved physics. However, the required knowledge is enormously smaller than that needed for building a new simulator from scratch, or even for modifying the core of an existing one.

In particular, and fortunately, EBM makes it possible to cope with the requirement just sketched with an amount of modelling culture accessible to many, and most important, with no need for the analyst to learn about the numerical model solution; we here contribute to putting the reader in the position of exploiting EBM for the above purpose.

II. Synopsis

As said, modern IC cooling systems give rise to multi-physics and inherently cyber-physical systems. To address such systems, it is necessary to touch several subjects. Given the wide audience we are addressing, each reader most likely has a differently distributed knowledge of the complete *panorama*. Achieving a consistent and streamlined presentation, suitable for all, is thus a challenging task. We provide in this section a minimal synopsis of the paper, showing what experts in one or another subject could quite safely skip, at least on a first reading.

- Section III provides a minimum of general modelling background, introducing in particular the idea of *declarative* – as opposed to *imperative* – model, and sets the required terminology. We think everybody should read this, at least to prevent misunderstandings.
- Section IV introduces EBM *by example*, to make the reader capable of mastering the basics enough to follow the rest of the treatise; references are provided for a further, more structured study. Experts of EBM can safely skip this section.
- Section V comparatively discusses declarative and imperative modelling, thereby providing evidence that in MPSoC cooling the two approaches need combining. Few ideas here would look new to EBM-proficient people. However, we consider it useful not to skip the section, at least for the MPSoC-related considerations.
- Section VI reviews the encountered physical phenomena, and how to describe them as differential equations. Experts of first-principle modelling – basically for thermo-hydraulic systems – could in principle go directly to the next section. Nevertheless, in this section we are also introducing some domain-specific approximations, and these may require one to use the section as reference for the following ones.
- Section VII explains how to describe the thermal phenomena that occur in MPSoC cooling systems using EBM tools. It also discusses how to handle spatial discretisations in an optimised way, a matter that could be of interest also to those already proficient in EBM modelling.
- Section VIII describes how Modelica models of cooling systems, typically built with the said library, can be connected to 3D-ICE — and potentially, to any external simulation code that can be endowed with an FMI interface compatible with the shown one.
- Section IX illustrates some examples of cooling circuits built with the library, also covering the interfacing with 3D-ICE. We suggest the interested reader to employ the presented models as a basis for his/her own first ones.
- Section X concludes the paper with some brief considerations and sketches out future developments and research activities.

We wish that the community will contribute to the development process we started, and consequently that the Modelica library we are presenting will foster the growth of an ecosystem for sharing modelling experiences and results.

For this reason, as well as to allow the reader to reproduce all of our experiments and to build on our models, we are releasing the library – both for standalone use and connectable to 3D-ICE with the provided interface – as free software, under a 3-clause BSD licence. On the same front, though the library should work with any Modelica tool, we developed and tested it with the free and open source translator OpenModelica [14], available at https://www.openmodelica.org, so that no barriers exist to using and sharing.

The library and the associated material can be downloaded at https://github.com/looms-polimi/computer_cooling/[1].

III. Modelling background

The word "model" has a wide range of meanings, and more must be added if we also embrace physical or cyber-physical simulation, so we first need to set out terminology. In the field of ICs and their heat dissipation systems as addressed herein, for *model* we mean

> *a mathematical description of a physical or cyber-physical system, aimed at* simulating *its behaviour over time under prescribed initial conditions and exogenous* stimuli.

When one needs to stress that a model replicates the transient behaviour and not only the steady-state conditions of a system, that model is called *dynamic*. In this work all models are dynamic, however, so hereinafter we drop the adjective.

[1]The library is continuously developed; we suggest the interested reader to clone the repository via `git` and check/download updates with `git pull`.

A. First-principle and data-based models

A model is *first-principle* when written based on the laws of physics, plus possibly some generally validated empirical correlations. It is conversely *data-based* when built by observing data recorded on the physical system, and suitably correlating them over time so as to replicate the input-to-output relationships in the modelled object without any knowledge or hypothesis about the physics in between. The parameters of a first-principle model almost always have a direct physical meaning (such as masses, specific heats, exchange coefficients and so forth) while those of data-based models hardly ever admit any such interpretation. Between the two *extrema* just mentioned there are many "mixed" modelling paradigms, for completeness, but a taxonomy of these is not in the scope of this work; a concise discussion is reported in [15].

When models are used to design some new physical equipment there is obviously no recorded data yet. Hence, one has to stick to the first-principle setting, as we dominantly do in this work; we shall therefore drop the "first-principle" attribute as well.

B. Declarative and imperative models

In our context as just sketched, a model starts out as a set of *dynamic balance equations* – e.g., the derivative with time of the mass in a volume equals at any instant the sum of the flow rates through its boundary – and *algebraic physical correlations*, like for example the Colebrook one to relate the pressure drop across a duct and the flow rate through it. Mathematically, the result is a DAE (Differential and Algebraic Equations) system, in general nonlinear and in some cases even discontinuous.

We call such a model *declarative*, as it holds all the information needed to simulate the modelled system but cannot be used *as is* to compute its behaviour — strictly speaking unless the DAE system can be solved analytically, but for models of engineering interest this is never the case.

A model that can be used for computing the system behaviour – i.e., with a slight simplification acceptable here, a solution algorithm for the DAE system – is conversely said to be *imperative*, as it can be unambiguously turned into a set of computer instructions to run.

The *Declarative-to-Imperative* (D2I) translation of a model is thus a necessary step for its simulation. The D2I translation is in general a complex and potentially critical process, however, hence it plays a prominent role in the following discussion.

C. Monolithic and modular declarative models

For simulating a system, its DAE model needs solving as a whole. Fortunately, however, this does not mean that the DAE must be *written* by the human as a single, comprehensive system, that is, be *monolithic*.

On the contrary, to tame the complexity of most systems it is far more convenient to build a (declarative) model by assembling *components*, that in turn may be the composition of other components in a hierarchical manner. We call models built this way *modular*.

In a modular modelling context, components must have *interfaces* so that the analyst can compose them together.

D. Causal and a-causal declarative models

We say that a model is *causal* when its interface with the outside consists of *inputs* and *outputs*: knowing the inputs (plus the internal states, as we deal only with dynamic models) is sufficient to know the outputs.

It is important to avoid confusing "causal" with "imperative". For example, a continuous-time transfer function model is causal – because it has an input and an output – but is still declarative until the choice of a numeric integration method turns it into an imperative procedure to compute state and output. Causal models must be *closed*, i.e., they must have as many equations as unknowns.

Conversely, a model is *a-causal* when its variables are not inputs or outputs *per se* but assume either role when the model is connected to others. For example, a resistor of resistance R is ruled by Ohm's law $v = Ri$; if it is connected in parallel to a voltage generator then voltage v is prescribed (hence it is the input) and current i is the output, while the opposite occurs if the resistor is in series to a current generator. A-causal models need not be closed: for example, the above resistor model has two unknowns (v and i) and one equation. In an a-causal model, the equations that specify its behaviour independently of how that model is connected to any other (in the resistor case, $v - Ri = 0$ for maximum clarity) are called the *constitutive equations*.

The interface of an a-causal model is made of *ports*, not inputs and outputs. Ports naturally lend themselves to represent physical connection points, like the pins of a resistor. They carry *variables*, that can be of two kinds: *effort* variables, that make sense with respect to a reference or as the difference between two points (such as voltage or temperature) and *flow* variables, that conversely make sense through a surface (such as current or heat rate). There is a third kind of variables named *stream*, but this is better introduced later on.

When connected, ports generate *connection equations*. These equations state that all their effort variables with the same name (think of the voltages in a set of pins soldered together) are equal, while all their flow variables with the same name (think of the currents in the same set and with uniform convention, e.g., positive if entering the pin) sum to zero. Together with the constitutive equations, the connection ones give rise to a closed compound model.

IV. Minimal introduction to EBM

This section is devoted to a nutshell-size and operational introduction to EBM. We employ to this end a simple example, with which we also start introducing the Modelica syntax. In the example we refer to an electrical case for simplicity, as such a system is most likely familiar for the reader. The goal is first to introduce the core modelling concepts required in this research. Of course, starting from the next section, we will switch to thermal models.

Coming to the example, we want to model a simple circuit made of step voltage generators, conductors, capacitors

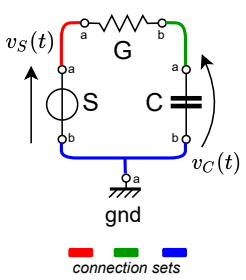
and ground. Listing 1 defines the required components from scratch, in declarative form. For completeness, we have to say that all the components in Listing 1 are already available as part of the Modelica Standard Library (MSL for short), that we shall mention again later. For a complete and self-contained explanation, however, at line 1 we here took from the MSL just the definition of physical quantities – voltage, current and so on – and their SI units of measurement (UoM).

```
1 import Modelica.SIunits.*; // type and UoM definitions
2 f
3 connector pin "Electric pin"
4        Voltage v "voltage";
5   flow Current i "current, positive if entering";
6 end pin;
7
8 partial model TwoPin "Generic two-pin component"
9   pin     a,b;
10  Voltage v;
11  Current i;
12 equation
13  a.i+b.i = 0; // current balance
14  a.v-b.v = v; // voltage across
15  a.i     = i; // current through (utiliser convention)
16 end TwoPin;
17
18 model Conductor "Ideal conductor"
19  extends   TwoPin;               // inherit base class
20  parameter Conductance G = 1e-3; // default value
21 equation
22  0 = i - G*v;                    // add Ohm's law
23 end Conductor;
24
25 model Capacitor "Ideal capacitor"
26  extends   TwoPin;
27  parameter Capacitance C     = 1e-6;
28  parameter Voltage     v_ini = 0;    // initial voltage
29 equation
30  0 = i - C*der(v);                   // diff. equation
31 initial equation
32  0 = v - v_ini;                      // initialise
33 end Capacitor;
34
35 model StepVoltage "Ideal step voltage generator"
36  extends   TwoPin;
37  parameter Voltage V0     = 0  "v before step";
38  parameter Voltage V1     = 10 "v after step";
39  parameter Time    t_step = 0  "time of step";
40 equation
41  v = if time<t_step then V0 else V1;
42 end StepVoltage;
43
44 model Ground "Ideal ground"
45  pin a;
46 equation
47  a.v = 0;
48 end Ground;
```

Listing 1: EBM introductory example – components.

Note the presence of *inheritance*, typical of object-oriented languages like Modelica: lines 8–16 define a generic component with two pins, and then this is specialised to be conductor, capacitor and voltage generator. By just inheriting (in Modelica, `extending`) `TwoPin` and adding the convenient v to i relationship, one can obtain resistor, inductor, diode, and so forth. Note also (line 41) the availability of *conditional equations*. Finally, line 22 could obviously be written as `i=G*v`, and analogously for lines 30 and 32, but we wanted to stress that component models contain equations and not assignment statements.

The Modelica listing on the right in Figure 1 shows how the defined components are used to build the simple circuit on the left, also overriding some parameter defaults. One can of course insert more occurrences of the same component, each with its own parameter values. The `connect` statements at lines 7–10 join connectors (pins) into three *connection sets*. As said, a set of $n \geq 2$ connectors generates n equations per effort variable to set all its values equal, and one per flow variable to set the sum of its values to zero. For example, the blue connection set (of 3 pins) generates




```
1 model Circuit_01
2   StepVoltage S(t_step=0.1);
3   Capacitor   C(C=1e-5);
4   Conductor   G(G=1e-4);
5   Ground      gnd;
6 equation
7   connect(S.a,G.a);   // red
8   connect(G.b,C.a);   // green
9   connect(S.b,gnd.a); // blue
10  connect(C.b,gnd.a); // blue
11 end Circuit_01;
```

Fig. 1: EBM introductory example – a complete model.

Entity	Variables introduced	Equations introduced	
S	6 $(a.v, a.i, b.v, b.i, v, i)$	4 (1+3 from `TwoPin`)	CONST
G	6 $(a.v, a.i, b.v, b.i, v, i)$	4 (1+3 from `TwoPin`)	
C	6 $(a.v, a.i, b.v, b.i, v, i)$	4 (1+3 from `TwoPin`)	
gnd	2 $(a.v, a.i)$	1	
Running total	20	13	
red set	none	2	CONN
green set	none	2	
blue set	none	3	
Total	20	20	

TABLE I: EBM introductory example – variables and equations (CONSTitutive and CONNection).

$$\begin{aligned} S.b.v &= gnd.a.v, \\ C.b.v &= gnd.a.v, \\ S.b.i + C.b.i + gnd.a.i &= 0. \end{aligned} \tag{1}$$

An open pin would be treated as a connection set with $n = 1$, resulting in the one equation $i = 0$ without any inconsistency.

The overall equations/variables balance for the model of Figure 1 is given in Table I. As can be seen, the individual component models are not closed, while thanks to the connection equations the compound one is.

Though we have just scratched the surface of EBM, and Modelica in particular, we can observe that constitutive equations – besides being in declarative form – do not depend on connections, while connection equations do not depend on the behaviour of the connected models. This allows us to point out a few relevant facts for the following discussion.

- There is a clear interface/behaviour separation, hence models are interchangeable as long as the interface (i.e., the connector structure) is preserved. This is particularly useful when the detail of a model (or part of a model) needs tailoring for the simulation study to conduct.
- Viewed differently, the separation is between internal behaviour and boundary conditions. A model is written independently of how it will be connected to others. Together with the declarative approach, this makes mod-

els resemble very closely the way they appear, e.g., as equations in a textbook.

V. Declarative and imperative pros/cons

In Section III-B we anticipated the importance of the D2I translation process. We now revisit the matter to evidence pros and cons of declarative and imperative modelling, coming at the end of this section to set the focus on ICs and their cooling systems. To do so, we need to introduce some details on how declarative models are translated into imperative ones by an EBM language automated translator — or how a modeler would have to translate models by hand if not using EBM tools. To this end we consider and discuss a couple of D2I translations, here too referring to simple purposed cases.

A. D2I – Example 1

In this example we consider a one-dimensional heat transfer problem in a solid, namely a rod heated at one side by a prescribed power $P_p(t)$ – t is time – and connected to a prescribed temperature $T_p(t)$ on the other side, the lateral surface being adiabatic. We denote by S the rod uniform section, by L its length, and by ρ, c, λ its constant density, specific heat and thermal conductivity.

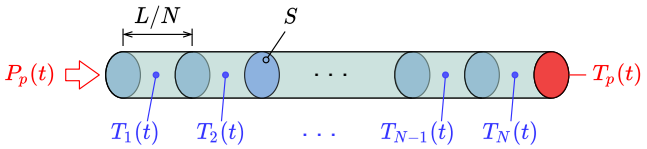


Fig. 2: D2I example 1 – the modelled system.

Approximating the continuous rod with a sequence of volume lumps, each one with its own temperature and exchanging heat with the previous and the following one, we readily get to the declarative model schematised in Figure 2 and written as the system of differential equations

$$\begin{cases} C\dot{T}_1(t) = P_p(t) - G\,(T_1(t) - T_2(t)) \\ C\dot{T}_i(t) = G\,(T_{i-1}(t) - 2T_i(t) + T_{i+1}(t)) \qquad i = 2\ldots N-1 \\ C\dot{T}_N(t) = G\,\big(T_{N-1}(t) - 3T_N(t) + 2T_p(t)\big) \end{cases} \tag{2}$$

where the dot indicates derivative with time, N is the number of volumes (or *lumps*) into which the rod is divided, T_i is the temperature (assumed spatially uniform) of the i-th lump, and

$$C = \rho c SL/N, \quad G = \lambda SN/L \tag{3}$$

are respectively the heat capacity of one lump and the centre-to-centre inter-lump thermal conductance. Since (2) is linear in T_i, P_p and T_p, we can write it in the compact matrix form

$$\dot{T}(t) = AT(t) + Bu(t) \tag{4}$$

where

$$T(t) = \begin{bmatrix} T_1(t) \\ \vdots \\ T_N(t) \end{bmatrix}, \quad u(t) = \begin{bmatrix} P_p(t) \\ T_p(t) \end{bmatrix} \tag{5}$$

and

$$A = \frac{G}{C}\begin{bmatrix} -1 & 1 & 0 & \cdots & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & -2 & 1 \\ 0 & \cdots & \cdots & 0 & 1 & -3 \end{bmatrix}, \quad B = \frac{1}{C}\begin{bmatrix} 1 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 2G \end{bmatrix}. \tag{6}$$

A possible way to translate (4) into imperative code (there are many but in this example we stick to one for simplicity) is to decide a time step h for computing the evolution of the *state variables* (those under time derivative) and replace all time derivatives with incremental ratios over h. We can do this explicitly or implicitly, that is, assuming either of the two approximations

$$\frac{x(kh) - x((k-1)h)}{h} \approx \begin{cases} \dot{x}((k-1)h) & explicit \\ \dot{x}(kh) & implicit \end{cases} \tag{7}$$

whatever x is, where the integer k counts the time steps at which the solution is computed. Writing for compactness $x(k)$ to mean $x(kh)$, again whatever x is, the two approximations in (7) respectively transform (4) into

$$\begin{aligned} T(k) &= (I + hA)T(k-1) + hBu(k) && expl. \\ T(k) &= (I - hA)^{-1}T(k-1) + (I - hA)^{-1}hBu(k) && impl. \end{aligned} \tag{8}$$

Based on this, a possible imperative model is provided by Algorithm 1. Observe that in this example it is reasonably easy to carry out the translation once and leave the task of specialising the model for a specific rod to a configuration file with physical parameters and the explicit/implicit choice. Analogously, initial conditions for a particular run can reside in the configuration file as well, and the inputs $u(k)$ can be acquired either from a file or by running some other simulation algorithm synchronously.

Summing up, as long as the problem to study concerns a rod under the stated conditions – we shall call this the *coverage* of the dynamic model we just wrote – the analyst can use that model by (i) compiling configuration, initialisation and possibly input files, and if needed (ii) writing another algorithm to suitably govern some inputs (for example, modulate T_p so that none of the T_i ever exceeds a threshold in the face of a time-varying P_p). *Said otherwise, the analyst needs no knowledge of the internals of the model, nor of the underlying principles. Also*, and most relevant for the following discussion, the introduced matrix compact form makes the D2I translation just parametric in the number of lumps: the matrices change in size, but their form is still the same.

B. D2I – Example 2

We now consider two resistors $R_{1,2}$ in parallel subjected to a prescribed voltage $v(t)$, that dissipate heat by Joule effect toward an air duct where R_2 comes downstream, hence receiving air already heated by R_1. The modelled system is illustrated in Figure 3.

We denote by $C_{t1,2}$ the thermal capacities of the two resistors, by c_a the specific heat of air, and by $G_{t1,2}$ the resistors-to-air thermal conductances; for simplicity we assume all these

Algorithm 1: D2I example 1 – imperative model.

```
   /* Problem acquisition & setup                    */
1  read L, S, N, ρ, c, λ, h, explicit from config file;
2  read initial values for T_i from init file;
3  compute matrices A and B as per (6);
4  if explicit then
5  |  A_discrete_time ← I + hA;
6  |  B_discrete_time ← hB;
7  else
8  |  A_discrete_time ← (I − hA)^−1;
9  |  B_discrete_time ← (I − hA)^−1 hB;
10 end
   /* Initialisation                                 */
11 k ← 0;
12 T_previous ← initial values for T_i;
   /* Simulation                                     */
13 while simulation not finished do
14 |  t ← kh;
15 |  acquire or compute u(k);
16 |  T ← A_discrete_time T_previous + B_discrete_time u(k);
17 |  write t to output file as t(k);
18 |  write T to output file as T(k);
19 |  T_previous ← T;
20 |  k ← k + 1;
21 end
```

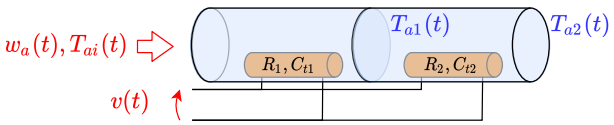


Fig. 3: D2I example 2 – the modelled system.

physical parameters constant, but we account for resistance variations due to temperature, i.e., we write

$$R_{1,2}(T_{1,2}) = R_{01,2}\big(1+\alpha_{1,2}(T_{1,2}-T_{01,2})\big) \tag{9}$$

where $T_{1,2}$ are the resistor temperatures, $\alpha_{1,2}$ their temperature coefficients, and $R_{01,2}$ the resistance values at the reference temperatures $T_{01,2}$. We neglect heat storage in the air, assume the duct walls to be adiabatic, and indicate with $w_a(t)$ and $T_{ai}(t)$ the air mass flow rate and inlet temperature, respectively. This said, we can express the time derivatives of T_1 and T_2 and then obtain the imperative model by approximating these with incremental ratios as we did above.

For this example, however, we do not really need to perform the D2I translation. It is enough to look at the expressions of $\dot{T}_1(t)$ and $\dot{T}_2(t)$, that respectively read

$$\dot{T}_1(t) = \frac{1}{C_{t1}}\left(\frac{v^2(t)}{R_{01}\big(1+\alpha_1(T_1(t)-T_{01})\big)} + G_{t1}\left(T_{ai}(t)-T_1(t) + \frac{v^2(t)}{c_a w_a(t)\big(R_{01}\big(1+\alpha_1(T_1(t)-T_{01})\big)\big)}\right)\right) \tag{10}$$

and

$$\dot{T}_2(t) = \frac{1}{C_{t2}}\left(\frac{v^2(t)}{R_{02}\big(1+\alpha_2(T_2(t)-T_{02})\big)} + G_{t2}\left(T_{ai}(t)-T_2(t) + \frac{v^2(t)}{c_a w_a(t)\big(R_{01}\big(1+\alpha_1(T_1(t)-T_{01})\big)\big)} + \frac{v^2(t)}{c_a w_a(t)\big(R_{02}\big(1+\alpha_2(T_2(t)-T_{02})\big)\big)}\right)\right), \tag{11}$$

to make two immediate considerations. First, the obtained expressions are nonlinear, hence the path to the imperative code will be inherently more complicated and error-prone than it was in Example 1 above. Second, and again most relevant, changing the number of cascaded "resistor in duct" elements changes the *form* of the equations, not only the size of some matrix: one just needs to observe (10) and (11) to conclude that the derivative of each temperature depends on that temperature and on all the upstream ones in the air duct. Also, the derivative expressions would change significantly should one for example insert a resistance in the electric supply line between R_1 and R_2 instead of assuming them perfectly in parallel or place them side by side (instead of downstream one another) in the air duct. Summing up, then, one could still get to an algorithm that can be used by just compiling configuration files, but this would entail accepting a coverage that is narrow indeed. If the analyst has to experiment with a variety of solutions trespassing the said coverage, the D2I translation must be re-done each time. This can be effort-heavy, and apparently requires knowledge of both the modelling principles to apply, and how to solve the obtained equations numerically.

C. Examples 1 and 2 the declarative way

1) Example 1: As shown in Listing 2, making the rod a Modelica component and using it to assemble an example model is quite straightforward once the `HP` (heat port) connector is defined; notice the similarity to the electric pin. Notice also the natural separation between the models of the rod and of the boundary conditions, incidentally.

2) Example 2: Besides nonlinearity, the presence of a moving fluid gives rise in example 2 to the transport of thermal properties with the fluid itself. This requires some equations to be conditional with respect to the direction (i.e., the sign) of flow rates. Most typically, in an energy equation, the flow rate through a volume boundary port must be multiplied by the enthalpy inside the volume if the fluid is exiting, and by the mix of enthalpies appearing at the port outside if the fluid is entering. The management of this complexity source is offered in Modelica by declaring transported variables as `stream` in a connector with one `flow` variable. Then, a model using that connector has to assign to each `stream` variable the value it takes inside the component, and when that variable appears in a balance equation, to apply to it the `actualStream` operator; the tool will generate all the required conditional equations transparently.

This capability is exploited in Listing 3, which also employs some of the electric components defined in Listing 1. Thanks

```
import Modelica.SIunits.*; // type and UoM definitions

connector HP "Heat port"
       Temperature T      "temperature";
  flow Power       Q_flow "thermal power, + if entering";
end HP;

model LumpedRod "1D lumped solid rod"
  HP          a,b;
  parameter   Length               L      = 0.1;
  parameter   Area                 S      = 1e-4;
  parameter   Density              ro     = 7600;
  parameter   SpecificHeatCapacity c      = 450;
  parameter   ThermalConductivity  lambda = 45;
  parameter   Integer              lumps  = 10;
  parameter   Temperature          T_ini  = 293.15;
  Temperature T[lumps](each start=T_ini);
protected
  final parameter HeatCapacity        C=ro*c*S*L/lumps;
  final parameter ThermalConductivity G=lambda*S*lumps/L;
equation
  C*der(T[1])     = a.Q_flow-G*(T[1]-T[2]);
  a.T             = T[1]+2*G*a.Q_flow;
  for i in 2:lumps-1 loop
      C*der(T[i]) = G*(T[i-1]-2*T[i]+T[i+1]);
  end for;
  C*der(T[lumps]) = G*(T[lumps-1]-T[lumps])+b.Q_flow;
  b.T             = T[lumps]+2*G*b.Q_flow;
end LumpedRod;

model PrescribedT
  HP        a;
  parameter Temperature T=293.15;
equation
  a.T = T;
end PrescribedT;

model PrescribedQ
  HP        a;
  parameter Power Qout=1;
equation
  a.Q_flow = -Qout; //exiting must be negative
end PrescribedQ;

model Example
  PrescribedQ PQ(Qout=10);
  LumpedRod   rod(lumps=20);
  PrescribedT PT;
equation
  connect(PQ.a,rod.a);
  connect(rod.b,PT.a);
end Example;
```

Listing 2: D2I example 1, the declarative way.

to the manipulation capability already mentioned as a peculiarity of EBM tools, the `Example` model shows how simple and fast it is to mode a cascade of an arbitrary number of "resistor in duct" elements — apparently, an almost prohibitive (or at least much more time-consuming) task to carry out in imperative form directly.

D. Abstracting with an eye on IC cooling

We start from some general facts, that the examples above should have clarified. First, as EBM tools allow to write declarative models, the analyst is relieved from the D2I burden. This burden can be high, and in general must be re-incurred every time the structure of the model is modified.

On the other hand, if one accepts to do the D2I translation manually, most often the obtained imperative code will be significantly more efficient than that generated by a declarative model translated by an EBM tool. We do not explain the reasons here as this would be quite long and inessential for our purposes; the interested reader can refer, e.g., to [16].

```
import Modelica.SIunits.*; // type and UoM definitions

connector pwh "port for fluid flow"
           Pressure         p "pressure";
    flow   MassFlowRate     w "mass flow";
    stream SpecificEnthalpy h "transported with w";
end pwh;

model ResistorInDuct
  pwh         air_a,air_b;
  pin         a,b;
  parameter   SpecificHeatCapacity ca    = 1020;
  parameter   Resistance           R0    = 1;
  parameter   Real                 alpha = 0.005;
  parameter   Temperature          T0    = 293.15;
  parameter   HeatCapacity         Ct    = 10;
  parameter   ThermalConductance   Gt    = 5;
  parameter   Temperature          T_ini = 293.15;
  Temperature T(start=T_ini);
  Temperature Ta;
  Resistance  R;
  Power       Pe,P2a;
equation
  air_a.p - air_b.p = 0; // no pressure drop
  air_a.w + air_b.w = 0; // no mass storage
  // air energy balance (static, no storage)
  0         =  air_a.w*actualStream(air_a.h)
               +air_b.w*actualStream(air_b.h)
               +P2a;
  P2a       = Gt*(T-Ta); // power to air
  air_a.h   = ca*Ta;     // stream management
  air_b.h   = ca*Ta;
  // electric equations
  a.i+b.i   = 0;
  a.v-b.v   = R*a.i;
  R         = R0*(1+alpha*(T-T0));
  Pe        = (a.v-b.v)*a.i;
  // heating resistor energy balance
  Ct*der(T) = Pe-P2a;
end ResistorInDuct;

model PrescribedpaTa "prescribed air p and T"
  pwh                          air_a;
  parameter Pressure           p = 101325;
  parameter Temperature        T = 293.15;
  parameter SpecificHeatCapacity c = 1020;
equation
  air_a.p = p;
  air_a.h = c*T;
end PrescribedpaTa;

model PrescribedwaTa "prescribed air w and T"
  pwh                          air_a;
  parameter MassFlowRate       w = 1e-5;
  parameter Temperature        T = 293.15;
  parameter SpecificHeatCapacity c = 1020;
equation
  air_a.w = -w;
  air_a.h =  c*T;
end PrescribedwaTa;

model Example
  parameter Integer n=5 "cascaded duct elements";
  PrescribedwaTa    src;
  ResistorInDuct    rids[n];
  PrescribedpaTa    snk;
  StepVoltage       S;
  Ground            gnd;
equation
  connect(src.air_a,rids[1].air_a);
  for i in 2:n loop // ducts cascaded to one another
      connect(rids[i-1].air_b,rids[i].air_a);
  end for;
  connect(rids[n].air_b,snk.air_a);
  for i in 1:n loop // all resistors in parallel
      connect(rids[i].a,S.a);
      connect(rids[i].b,gnd.a);
  end for;
  connect(S.b,gnd.a);
end Example;
```

Listing 3: D2I example 2, the declarative way.

Now let us focus on ICs, and reconsider Examples 1 and 2 above. Example 1 is in fact a toy version of a 3D chip thermal model (1D geometry instead of 3D, only one heat source, constant material properties, and so on). Building on Algorithm 1, it is possible to create an extremely fast imperative model, realistic enough for a real chip, and that can be adapted to any case at hand by just compiling configuration files; the D2I translation can still be costly but is needed only once. With further additions not relevant at this point, such a configurable model is exactly what 3D-ICE and similar tools provide.

Example 2 is an equally toy version of an air cooling network inside a rack (only one stream, no pressure drops described, no air humidity, and so on). In such a context, for the imperative approach it would be evidently prohibitive to follow the variety of possible cooling network (thus model) topologies — let alone, for example, extensions to fluid phase changes, electro-mechanical impelling, or multiple interacting fluid circuits. Here, as the example should have proven, there is simply no "writing a general code and then specialising it with configuration files"; the D2I translation can be even more costly than in Example 1, owing to nonlinearities, and is needed for every case.

The so outlined *scenario* explains why EBM – that conversely can manage heterogeneous model complexity by suitably abstracting modular *declarative* components – needs bringing into play for modelling modern IC cooling. Of course, doing so requires a bit more consciousness of modelling principles than just compiling configuration files, but first this is inevitable given the variety of cases to address, and then – *as the burden of manual D2I translation is avoided* – it can be made accessible to many, as we are going to discuss in the following.

VI. MPSoC/IC cooling – modelling principles

In this section we define and discuss the principles on which a modelling solution for the cooling of modern ICs – like MPSoCs – must be grounded, and how to apply those principles to the addressed domain. The reader interested in more details about the used modelling methodologies can refer, e.g., to [17]–[19] or many analogous works. In the following we just report a few references relative to the considered phenomena and their dynamic description.

A. Substances

Any first-principle equation has to do with properties of materials. In MPSoC cooling these include the following.

- Solids, whose thermodynamic state in the conditions of interest for us is fully described by their temperature T, and that are physically characterised by a density ρ, a specific heat c and a thermal conductivity λ that one can assume constant in the said conditions [20].
- Single-phase (subcooled) liquids, whose state is described by their pressure p and specific enthalpy h, and whose physical properties (density, specific heat, thermal conductivity) can be assumed constant in any case of interest [21].
- Gases that behave (almost) ideally, i.e., whose state is described by their pressure p and temperature T (or equivalently, specific enthalpy h) and whose behaviour is ruled by

$$\frac{p}{\rho} = R^* T, \quad e = c_v T, \quad h = e + \frac{p}{\rho}, \tag{12}$$

  where ρ is density, e is internal energy, and R^*, c_v are respectively the considered gas constant and its constant-volume specific heat [21].
- Phase-transitioning species such as refrigerating fluids, not treated in this tutorial owing to complexity and space limits. For these the same modelling considerations given above apply, however, provided relationships are available to compute their properties based on pressure p and specific enthalpy h. Notable examples are the water-steam tables, included in the MSL, or the `ExternalMedia` package [22] to interface Modelica with external fluid property calculation software Like, e.g., REFPROP [23] and CoolProp [24].
- Fluid mixtures, most notably moist air, not treated herein either. These are handled in the same way above once analogous relationships are available (such as Mollier-like ones, to compute moist air properties based on pressure p, temperature T and absolute humidity X).

We now proceed to write the most important equations for the phenomena of interest: the hypotheses just made apply throughout, hence are not repeated for brevity.

B. Mass storage

Dynamic mass balances can refer to closed volumes filled with fluid (e.g., a pipe), to closed ones not completely filled (e.g., a gas pressuriser for a liquid circuit), or to open ones (e.g., a vented tank). Here we only treat the completely filled case with liquid and gas, and the vented case with liquid [25].

In the filled liquid case the equation is trivial, as the contained mass is constant. Denoting by w_i the generic mass flow rate through the i-th out of n_p volume port (e.g., the flanges of a pipe or of a header) the equation is therefore

$$\sum_{i=1}^{n_p} w_i = 0. \tag{13}$$

while pressure is determined by some of the other components of which the complete model is made.

In the filled gas case, denoting by V the volume, the contained mass is ρV, hence (recall that the dot means derivative with time) the equation reads

$$V\dot{\rho} = \sum_{i=1}^{n_p} w_i, \tag{14}$$

that recalling (12) becomes

$$\frac{V}{R^*}\frac{d}{dt}\left(\frac{p}{T}\right) = \sum_{i=1}^{n_p} w_i. \tag{15}$$

In (15) T can be considered constant or come from some thermal equation, see Section VI-E.

In the vented liquid case, the contained mass is represented by a level ℓ, so that denoting by A the base area of the volume (assumed cylindrical for simplicity) we get

$$M = \rho A \ell \tag{16}$$

whence

$$\rho A \dot{\ell} = \sum_{i=1}^{n_p} w_i \tag{17}$$

while pressure depends on level by the Stevinus law

$$p = \rho g \ell, \tag{18}$$

g being the gravity acceleration. Note that p in (18) is relative (or gage) pressure: if absolute pressure is needed one has to add the atmospheric one.

C. Mass transfer

The transfer of fluid mass can happen spontaneously owing to pressure differences like in pipes and valves, or be caused by some impelling mechanism like in pumps and fans [26]. In both cases the pressure drop Δp across the element is related to the mass flow rate w through it by an equation grounded in the conservation of momentum. In any situation relevant to us it is acceptable to neglect the effect of the fluid inertia, which allows writing algebraic equations in the form

$$\Delta p = \Delta p_0(\theta, u) + f(w, \theta, u) \tag{19}$$

where Δp_0 is a flow-rate-independent term due to external forces like gravity and/or the zero-flow-rate pressure difference in active elements like pumps, while function f accounts for friction losses and can take various forms (owing e.g. to laminar or turbulent flow) with $f(0, \theta, \cdot) = 0$; vector θ contains the physical parameters of interest, relative to the fluid, to the element in which the motion occurs (e.g., the flow coefficient of a valve or the static head of a pump) and sometimes to the installation conditions (e.g., the inlet-outlet height difference for a pipe). Input u may be present to represent a command when one exists (e.g., valve opening or pump speed set point).

D. Energy transfer with mass

A means to transport energy, of paramount importance in cooling, is by transporting mass [26]. When a mass stream enters or exits a volume, there are two effects: one is the energy contribution owing to the internal energy e of the entering or exiting fluid, the other is the (signed) mechanical work exerted by the stream on the fluid in the volume. This work can alter the pressure in the volume, thus be viewed in differential form as $pdv = pd(1/\rho)$ – where $v = 1/\rho$ is specific volume – which is often termed "pressure work", or maintain the fluid flow across the volume, thus be viewed as dp/ρ which is named "impelling work". Summing the two differentials we obtain $d(pv) = d(p/\rho)$, which explains why enthalpy is the natural thermodynamic variable to associate with energy transfers that occur by mass transfer.

Denoting by w the mass flow rate through a port and by h the enthalpy on the side of the port from which the fluid is coming, the energy rate at the port is therefore wh (with the sign dictated by the fluid direction as just noted).

E. Energy storage

Dynamic energy balances can refer to the same cases we classified mass balances with, and take the form

$$\frac{d}{dt}(Me) = \sum_{i=1}^{n_p} w_i h_i + \sum_{j=1}^{m_p} Q_i, \tag{20}$$

where M is the mass in the volume, e its specific energy, w_i and h_i are the exchanged flow rate and specific enthalpy at the i-th port (out of n_p) where energy is given or taken by transport of mass, see Section VI-D, and Q_i the exchanged power at the j-th port (out of m_p) where energy is given or taken without mass transport, see Sections VI-F and VI-G.

For solid elements M is ρV – hence it is constant – and e is cT, which makes temperature the natural state variable.

For the filled liquid volume case M is constant, but for the reasons in Section VI-D the natural state variable is h, hence e is expressed as $h - p/\rho$ (although the p/ρ term is largely dominated by e, that is cT, in any case relevant for us).

The situation is analogous for the filled (ideal) gas volume case, expressing M as ρV where ρ, p, e, h and T are related by (12). The energy balance can still be written in the form (20), however, thanks to the symbolic manipulation capabilities of Modelica tools.

For the open liquid volume case, finally, M is not constant, see (16), but here too (and for the same reason above) the balance can be written in the form (20), with $e = h - p/\rho$.

F. Spontaneous heat exchange

Heat flows spontaneously by conduction, convection and radiation (which we do not treat in this work owing to its low relevance in MPSoC cooling).

In our context, conduction is relevant only within solids because fluids move around, and energy transfers with mass make conduction within them negligible [27]. Also, in MPSoC cooling circuits the only solid elements are fluid containment ones, and for them conduction is only relevant in the direction orthogonal to the surface in contact with the fluid. The reason is again that fluids move around: considering for example a pipe wall as a sequence of annular elements, the heat transported from an element to the downstream one by the moving fluid by convection largely dominates the heat transferred by the former to the latter through solid-to-solid contact.

As a result, in cooling circuits we only have to do with one-dimensional conduction between two surfaces, in planar or cylindrical geometry. For the planar case we can neglect border effects and compute the thermal conductance G_t between two identical surfaces of area A, separated by a layer of thickness s of a material with thermal conductivity λ as

$$G_t = \lambda \frac{A}{s} \tag{21}$$

while for the cylindrical case, considering an annulus of length L, inner radius r_i and outer radius r_e, we get

$$G_t = \lambda \frac{2\pi L}{\log(r_e/r_i)}. \tag{22}$$

In both cases, denoting by T_1 and T_2 the two surface temperatures and by Q_{12} the conductive heat rate from surface 1 to surface 2, we have

$$Q_{12} = G_t(T_1 - T_2). \tag{23}$$

Convection is a more complex phenomenon, as it involves the motion of a fluid in contact with a solid. In particular, we need to distinguish *laminar* and *turbulent* flow. In a nutshell and thinking of a pipe, laminar flow is when fluid particles proceed aligned in the direction of the pipe axis, pushed by pressure difference and subject to mutual friction with the adjacent ones (or the wall). This happens at low speeds and results in a parabolic speed profile, with the fastest particles at the pipe centre. At higher speeds, the order of laminar flow breaks. Particles still move on average along the tube axis, but instead of proceeding pretty much like in parallel lanes they also continuously mix up with the neighbouring ones, resulting in an almost flat speed profile. As turbulence apparently helps giving or taking heat, owing to fast fluid particle turnover in the vicinity of the wall, turbulent convection is significantly more efficient than laminar one.

We thus compute the fluid to solid convective heat rate Q_{fs} referring to a bulk (average) fluid temperature T_f and a surface one T_s for the contacting solid, in the form

$$Q_{fs} = \gamma A(T_f - T_s) \tag{24}$$

where A is the contact surface and γ the convective heat transfer coefficient, computed from the fluid properties and motion conditions with correlations like the well-established Dittus-Bölter one. Notice that contrary to the conduction case γ is not constant; it depends on other variables in the system, most typically fluid speed and thus flow rate, making the phenomenon nonlinear.

G. Work-driven heat exchange

Heat pumps, which include Peltier elements, are machines that employ mechanical or electrical power to seemingly violate the second principle of thermodynamics by transferring heat from the colder to the hotter.

For simulating and evaluating a cooling system there is no need to represent the internal physics of such devices: it is more practical to describe them based on the nominal external characteristics provided by manufacturers. These are an efficiency and a Coefficient of Performance (COP), either constant or – more frequently and realistically – depending on the operating conditions.

Denoting by Q_c the heat rate taken from the cold side of the pump, by Q_h the heat rate released to the hot side and by W_p the net power to the pump, we define the COP as the ratio of the desired effect to the spent effort, i.e.

$$COP_{heat} = \frac{Q_c}{W_p}, \quad COP_{cool} = \frac{Q_h}{W_p}, \tag{25}$$

depending on whether the pump is devoted to heating or cooling; notice that $COP_{heat} = COP_{cool} + 1$, since obviously $Q_h = Q_c + W_p$. In our context we are always cooling, hence we drop the subscript and define $COP = Q_c/W_p$.

The above established, we can describe a heat pump as

$$\begin{aligned} Q_h &= Q_c + W_p \\ Q_c &= COP \cdot W_p \\ W_p &= \eta W \\ COP &= f_{COP}(T_h, T_c) \end{aligned} \tag{26}$$

where T_c and T_h are the cold and hot side temperatures, and f_{COP} a function – most frequently the interpolation of some measured points – to obtain the COP; η, in general reasonably constant, accounts for losses that cause the absorbed power W to be greater than W_p. The COP is limited by the Carnot efficiency, that *expressing temperatures in Kelvin degrees* for the cooling case reads

$$COP_{Carnot} = \frac{T_c}{T_h - T_c}. \tag{27}$$

Real heat pumps rarely achieve a COP above 40–50% of the Carnot one (a useful information for first-cut sizing). Moreover, notice that the COP – not only the Carnot one, clearly – diminishes as the pump temperature difference $\Delta T = T_h - T_c$ increases. This brings into play the heat exchangers connecting the pump to the cold and hot environments, as summarised in Figure 4.

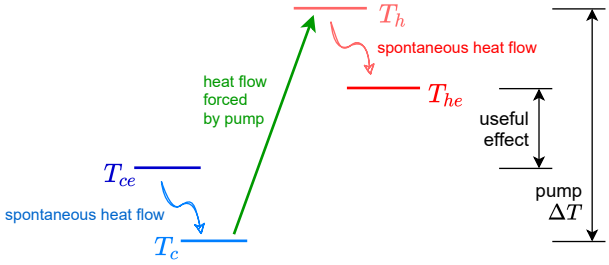


Fig. 4: Temperatures and flows in a heat pump.

Heat needs to first flow spontaneously – by conduction or convection – from the cold environment (the one to cool) at temperature T_{ce} to the cold side of the pump at T_c. Then, together with the absorbed energy, this heat will be transported to the hot side at temperature T_h, from which it will flow – here too spontaneously – to the hot environment at T_{he}. The useful cooling effect obviously occurs from T_{ce} to T_{he}, but if the thermal conductance relative to spontaneous flows is not large enough, this effect can result in a pump ΔT significantly higher than the effect itself, to the detriment of the COP.

VII. A Applying the presented concepts using EBM

In this section we show how to turn the principles and equations described in Section VI into functional EBM models for cooling system components. In this respect, it must be noted that modelling can be applied either to gain insight into existing systems, or to design entirely new ones. In this work, compared to our previous papers such as [9], we focus on presenting a modelling approach – also in the form of a support library – that can be applied to either case. Consequently, our library is modularised in generic components, that can be validated individually and then reused to compose models for arbitrary cooling systems.

A. Defining connectors

We already defined some connectors for the examples in Section V; we here re-collect the definitions for the reader's convenience, and add a few comments. From now on, we assume `Modelica.SIunits.*` imported in all listings.

```
/* For compatibility with other libraries we use the
   heat port defined in the MSL, namely in package
   Modelica.Thermal.HeatTransfer.Interfaces,
   where temperature is T and heat rate Q_flow
*/

connector pwh "port for fluid flow"
         Pressure         p "pressure";
  flow   MassFlowRate     w "mass flow";
  stream SpecificEnthalpy h "transported with w";
end pwh;

connector vHP "vector heat port"
  parameter Integer     n=3       "default size";
            Temperature T[n]      "T vector";
  flow      Power       Q_flow[n] "Q_flow vector";
end vHP;

connector mHP "matrix heat port"
  parameter Integer     nr=3          "default rows";
  parameter Integer     nc=3          "default columns";
            Temperature T[nr,nc]      "T matrix";
  flow      Power       Q_flow[nr,nc] "Q_flow matrix";
end vHP;
```

Listing 4: Connectors useful for modeling cooling circuits.

Vector and matrix heat ports serve when spatially distributed phenomena come into play, for example to represent the temperature profile along an exchanging duct, or the power distribution over a surface; this matter is addressed in Section VII-C.

B. Storage and transfer components

In this section, tightly related to the following Section VII-C, we introduce an important distinction. Although the storage and transfer of mass and energy take place contextually in both time and space, for an effective modularisation of models it is convenient to distinguish "storage" and "transfer" components and to give them a standardised structure, as depicted in Figure 5 and explained below.

1) Storage components: these contain dynamic balances of mass and energy, and correspond to a control volume in which the properties of the contained fluid are considered spatially uniform. Denoting by θ a vector containing the required physical parameters such as geometric dimensions and material constants, these components have the structure reported below, and along which the reader can build his/her new components.

- At lest one `pwh` connector, see Section VII-A; we denote by n the number of such connectors.
- If the containment boundary is not adiabatic, at least one heat port (scalar version of `vHP` in Section VII-A, with one T and one Q_{flow}); we denote by m the number of such connectors.
- Two equations to compute the contained mass M and the contained energy E, most frequently in the form

$$\begin{aligned} M &= \rho(p,h)V \\ E &= M\left(h - \frac{p}{\rho(p,h)}\right) \end{aligned} \tag{28}$$

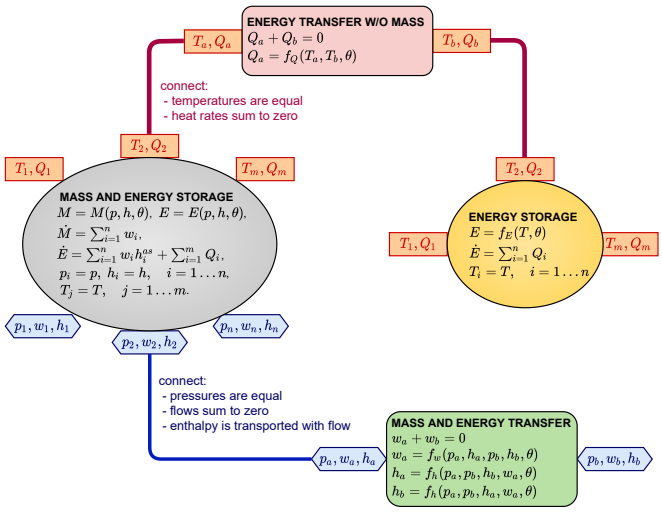


Fig. 5: Storage and transfer components and interconnections thereof.

where V is the volume and the other symbols have the known meaning; $\rho(p,h)$ represents the dependence of density on pressure and specific enthalpy as dictated by the model of the contained substance (in our context a single species).

- Two equations to compute the time derivatives of M and E, that is,

$$\begin{aligned} \dot{M} &= \textstyle\sum_{i=1}^{n} w_i \\ \dot{E} &= \textstyle\sum_{i=1}^{n} w_i \,\texttt{actualStream}(h_i) + \sum_{j=1}^{m} Q_{flow,j} \end{aligned} \tag{29}$$

where w_i is the flow rate at the i-th `pwh` connector and $Q_{flow,j}$ the heat rate at the j-th heat port; the Modelica clause `actualStream` takes the correct enthalpy (that of the contained flow or that presented from outside the connector) depending on the sign of each w_i.

- Equations to present at connectors the thermodynamic state (p,h) of the contained fluid, i.e.,

$$\begin{aligned} p_i &= p & i &= 1 \ldots n \\ h_i &= h & i &= 1 \ldots n \\ T_j &= T(p,h) & j &= 1 \ldots m \end{aligned} \tag{30}$$

Listing 5 illustrates (omitting unnecessary Modelica lines) how the above applies to describing a volume containing an ideal gas, with two `pwh` connectors `pwh_a`, `pwh_b` and one heat port `hp`.

C. Managing spatial discretisation

In the context we address, one often encounters variables with a spatial distribution: think for example of the pressure and temperature profiles along a pipe in a heat exchanger, or the temperature field in a silicon die. Such variables depend on both time and space, hence in the most general case on four independent coordinates, one temporal and three spatial. Moreover, their behaviour depends on transport phenomena, i.e., their value at a certain point in space is influenced by values at infinitely close other points. As a result, when spatially distributed variables come into play, models contain

```
...
  parameter              Volume              V;
  parameter              MolarMass           MM;
  parameter              SpecificHeatCapacity cv;
  parameter              Pressure            pstart;
  parameter              Temperature         Tstart;
  Pressure               p(start=pstart);
  Temperature            T(start=Tstart);
  Density                d;
  SpecificInternalEnergy e;
  SpecificEnthalpy       h;
  Mass                   M;
  Energy                 E;
protected
  final parameter SpecificHeatCapacity
        Rgas = Modelica.Constants.R/MM;
  final parameter SpecificHeatCapacity
        cp   = cv+Rgas;
equation
  p/d     =  Rgas*T;
  e       =  cv*T;
  h       =  cp*T;
  M       =  V*d;
  E       =  M*e;
  der(M)  =  pwh_a.w+pwh_b.w;
  der(E)  =  pwh_a.w*actualStream(pwh_a.h)
            +pwh_b.w*actualStream(pwh_b.h)
            +hp.Q_flow;
  pwh_a.p =  p;
  pwh_a.h =  h;
  pwh_b.p =  p;
  pwh_b.h =  h;
  hp.T    =  T;
...
```

Listing 5: Example component code – volume with ideal gas.

partial differential equations, where derivatives with time and with spatial coordinates appear.

Few modelling languages offer native support for partial derivatives, and when present, this support tends to be geared to a particular domain (a notable case is gPROMS [28] for the process industry). As such, in general it is the task of the model and library developer to take care of *spatial discretisation* to represent the distributions with a finite number of variables.

There are two main approaches to this problem, namely the finite-volume and the finite-element one. We stick here to the former, in which the region of space where variables are distributed is decomposed in volumes, where the value of the distributed variables is assumed uniform, and that exchange energy and possibly mass with the neighbouring volumes and/or the region boundaries. There is a lot of mathematics involved that we are not discussing in this paper; the interested reader can refer, e.g., to [29], [30], as well as to works like [31], [32] for the case of modern ICs.

Below we briefly treat the one-dimensional case, typically found in fluid elements as happens, e.g., for the temperature along a duct, and the three-dimensional case, which is typically seen when modelling temperature distributions in solids. The reason is that in ducts we can assume thermal properties to be constant over a surface orthogonal to the duct axis, whence the 1D case, and that we do not need to model three-dimensional fluid motion, whence the 3D case for solids only. The two-dimensional case is only found in *connectors* to describe contact surfaces.

1) One-dimensional case: The reader may notice that in fact we already used the finite-volume approach, thanks to its intuitiveness, when modelling the rod of Section V-A. In the more general case where fluid motion is present, like in pipes, one can resort to a scheme like the one in Figure 6.

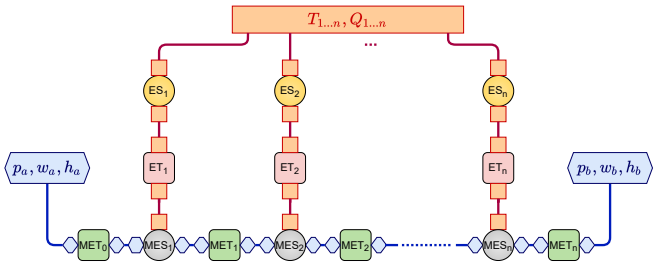


Fig. 6: One-dimensional exchanging fluid stream, modelled as a series of alternating storage and transfer components, with containment wall; see Figure 5 for the definition of component types and their roles.

In the shown example, a pipe is represented by a sequence of storage elements, summing up to the entire volume of the pipe, alternated with transfer elements to model the transport of fluid along the pipe. Figure 6 also comprehends the pipe wall, in the form of a sequence of (solid) storage elements with one side exchanging heat with the fluid, and the other exposed with a `vHP` connector to the outside environment. It is worth noticing that the structure in Figure 6 does not depend on the internals of the composing elements taken from Figure 5, hence being general with respect to the involved substances property calculation and to the used exchange correlations. This is another example of how EBM relieves the analyst from repetitive and error-prone modelling tasks.

2) Three-dimensional case: this is typically seen when modelling temperature distributions in solids, that adopting the finite-volume approach are decomposed into parallelepiped elements. For simplicity we discuss here the uniform grid case, i.e., all the elements have the same size: generalisation to structured grids is straightforward but would add further complexity and obfuscate the concepts being presented.

Though model optimisation is beyond the scope of this tutorial, we have to notice here that a naïve modelling approach would impact computational efficiency up to a relevant extent. We thus first show the most natural and human-readable way to proceed, then explain why with present Modelica tools this produces inefficient code, and finally suggest a (less readable) efficient modelling alternative, to be preferred until compilers evolve to close the evidenced gap.

The most human-readable approach to finite-volume 3D discretisation consists of first creating a model of the individual

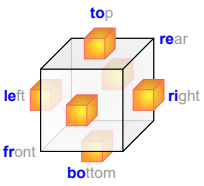


Fig. 7: Basic building block of a naïve spatial discretisation. Finite volume model with six thermal ports.

volume with connectors on each side, as shown in Figure 7, and then building a 3D array of such models, using `for` loops with `connect` statements. For the individual volume, denoting by T the temperature of one such element, by ℓ_x, ℓ_y and ℓ_z its dimensions along the three Cartesian axes, and by the subscripts `to`p, `bo`ttom, `le`ft, `ri`ght, `fr`ont and `re`ar the six heat ports HP_f for its faces, one gets to the model

$$\begin{aligned} \rho c(T)\dot{T} &= \textstyle\sum_{f=\{to,bo,le,ri,fr,re\}} HP_f.Q_{flow} \\ HP_f.T &= T + G_f HP_f.Q_{flow},\ f=\{to,bo,le,ri,fr,re\} \\ G_f &= \begin{cases} \lambda(T)\frac{\ell_x\ell_y}{0.5\ell_z}, & f=\{to,bo\} \\ \lambda(T)\frac{\ell_y\ell_z}{0.5\ell_x}, & f=\{le,ri\} \\ \lambda(T)\frac{\ell_x\ell_z}{0.5\ell_y}, & f=\{fr,re\} \end{cases} \end{aligned} \tag{31}$$

where c and λ are respectively the material specific heat and thermal conductivity, possibly temperature-dependent, while the density ρ is taken as constant. Centre to face conductances are here approximated assuming a planar geometry, which is acceptable for the typical purposes of our models; of course more precise approximations could be used without impairing the structure of the model and the consequent modularity.

Once the element is described as per (31), constructing a parallelepiped solid just amounts to connect elements via three sets of three nested `for` loops each, using `connect` statements to link volumes with their neighbours along the three axes. As an example, the following code shows the loops required for connection along the vertical axis.

```
for j in 1:M loop
 for k in 1:P loop
  for i in 1:N-1 loop
   connect(vol[i,j,k].bo,vol[i+1,j,k].to);
  end for;
 end for;
end for;
```

In principle the presented modeling strategy should be handled efficiently by Modelica compilers, however to date Modelica tools are not able to infer arrays of variables from arrays of components that contain those variables and thus perform optimisations across components. Naïve models such as the one just presented thus generate a large number of unnecessary algebraic equations arising from the interconnections of conductances at the faces of the finite volumes, thus resulting in a significant simulation overhead.

This issue can be overcome by structuring the 3D discretisation model as a *single component* containing a 3D array of state variables – temperatures in this case – and writing exchange equations referring directly to the state variables without involving the temperatures at the faces of the finite volumes. This approach has the joint effect of *a priori* reducing the total number of equations in the model and avoiding algebraic equations, both factors contributing to improved simulation speed.

The trade-off is the complexity of having to handle the solid boundaries explicitly. In fact, while the naïve finite volume model handled unconnected ports automatically, in this case we need to handle separately all possible cases where one or more of the volume faces is facing outwards of the solid.

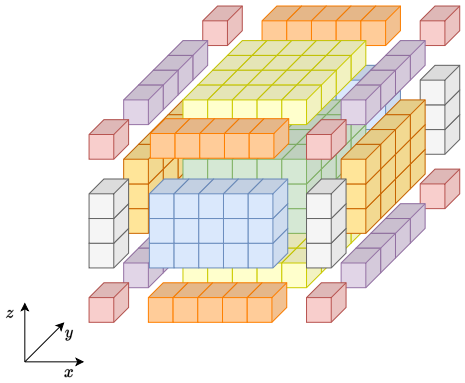


Fig. 8: Volume decomposition for optimised 3D spatial discretisation via uniform grid.

A graphical representation of all the different cases is shown in Figure 8, where the different element colours evidence the need for seven groups of `for` loops, plus eight individual equations for the solid vertices (that do not require loops). For example, denoting by N,M,P the number of elements along the three axes and by T the 3-dimensional array of volume temperatures, the equations for the inner volumes (green elements in Figure 8) in Modelica read

```
for i in 2:N-1 loop
 for j in 2:M-1 loop
  for k in 2:P-1 loop
   der(T[i,j,k]) = 1/C*(
     Gx*(T[i-1,j,k]-2*T[i,j,k]+T[i+1,j,k])
    +Gy*(T[i,j-1,k]-2*T[i,j,k]+T[i,j+1,k])
    +Gz*(T[i,j,k-1]-2*T[i,j,k]+T[i,j,k+1]));
  end for;
 end for;
end for;
```

As said, a current research line in the field of Modelica compilers aims to make them array-aware (see e.g. the research presented in [16]), so next-generation Modelica compilers will be capable of handling the human-readable alternative equally efficient, but until then, manual optimisation is required.

VIII. Interfacing with 3D-ICE

3D-ICE is a free and open source thermal simulator specifically designed for ICs. Its purpose is to model a silicon die, together with an optional heat spreader, through finite volume discretisation. The model is configurable in terms of material properties, dimensions, finite volume sizes and number of layers. Numerical integration of the resulting differential equations is performed using the implicit Euler method, optimised to efficiently produce high-resolution transient temperature maps.

To model arbitrary heat dissipation systems, we recently introduced in 3D-ICE support for co-simulation [9] with Modelica models through the FMI (Functional Mockup Interface) standard [12]; FMI provides a co-simulation interface that allows Modelica models to be exported as shared libraries, to be loaded by the 3D-ICE pluggable heat sink infrastructure.

A typical co-simulation workflow begins with the modelling of a heat dissipation system through the graphical user interface of a Modelica environment such as OpenModelica. Modelling is assisted by the use of Modelica libraries of components such as those provided as part of the 3D-ICE distribution, the library we are presenting as part of this paper, as well as the Modelica standard library. In order to be suitable for co-simulation, the Modelica model of the heat dissipation system will need to expose a specific interface that will be described shortly. Once the model is ready, co-simulation with 3D-ICE entails abandoning the graphical user interface of Modelica environments, and integrating with the 3D-ICE build system. We omit low-level details about this topic that can be found in the 3D-ICE documentation, but we remark that as part of the Modelica library we are releasing we provide a complete co-simulation example with 3D-ICE along with detailed instructions on setting up the co-simulation environment.

A. *Co-simulation interface*

```
partial model Heatsink
  parameter Temperature initialTemperature(fixed=false);
  parameter ThermalConductance cellBottomConductance;
  parameter Length bottomLength;
  parameter Length bottomWidth;
  parameter Integer bottomRows;
  parameter Integer bottomCols;
  HeatPort_a bottom[bottomRows, bottomCols];
end Heatsink;
```

Listing 6: Heat sink component interface for 3D-ICE interfacing

To be co-simulated with 3D-ICE, a Modelica model of a heat dissipation system, however complicated, needs to be wrapped into a single model component exposing the heat sink bottom surface to 3D-ICE by means of implementing the interface of Listing 6. As Modelica is an object-oriented language, the `Heatsink` interface is provided as part of 3D-ICE in the `HeatsinkBlocks` library and can simply be inherited by every heat sink model.

This interface is composed of 6 parameters, and a matrix of heat port connectors to model heat transfer between the chip and heat sink. The interface assumes that the heat sink bottom side is discretised using at least one layer of finite volumes, and requires to connect the heat port array to the *centre* of the volume cells of the (bottommost) layer. Additionally, the heat sink model must set the `cellBottomConductance` parameter to reflect the value of the thermal conductance between the centre and the bottom surface of the (bottommost) layer. The reason for not exposing the bottom surface directly through the matrix connector lies in the need for 3D-ICE to perform grid-pitch adaptation, a feature that will be discussed shortly. Additional parameters need to be set by the heat sink model to inform 3D-ICE of the bottom surface length and width, as well as the number of finite volumes the surface is discretised in.

It should be noted that the physical dimensions and number of volumes of the matrix heat port exposed by the Modelica model do not need to be -and most commonly is not- equal to the physical dimension and number of volumes of the silicon die/heat spreader modeled by 3D-ICE. For what concerns physical dimensions, a heat sink is usually significantly larger than a silicon die or a heat spreader. Also, when considering finite volume discretisation, the silicon die often needs to be simulated at a very fine level of discretisation to observe hot spots and the temperature of different functional units in the IC floorplan, while a heat sink can be simulated at a much coarser grid size. For this reason, 3D-ICE includes a grid pitch adaptation layer allowing to perform co-simulation between a chip and a heat sink of different sizes and with different discretisations. For what concerns different discretisations the mapping is completely transparent, while the size mapping allows specifying the chip location with respect to the sink bottom surface. Figure 9 shows a pictorial view of the grid pitch mapping across the co-simulation interface.

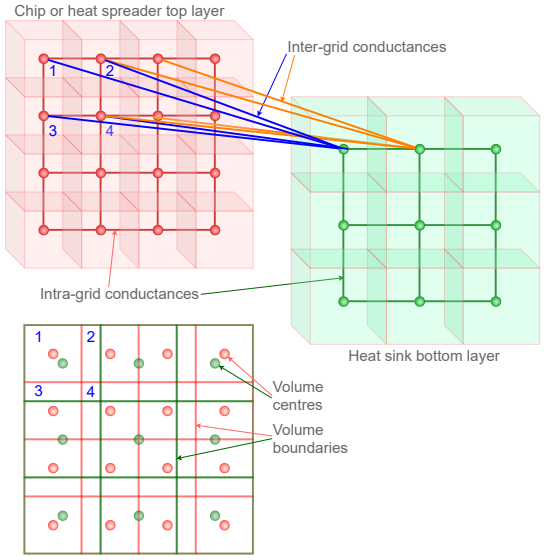


Fig. 9: Grid mapping across the co-simulation interface between Modelica and 3D-ICE; the four blue conductances 1–4 connect the top left green grid volume to the four orange grid volumes its facing surface overlaps with, as the orange ones do with the top centre green volume — and so forth.

The final parameter of the interface is the `initialTemperature`, representing the temperature of the heat sink at the start of the simulation. It should be noted that in Modelica, the keyword `fixed` does not mean constant, and in particular `fixed=false` applied to a parameter means it can be left without a value, and the actual value will be set externally before starting the simulation. In this case, 3D-ICE also has an initial temperature parameter, set through its configuration file. This feature is thus used to avoid the need to write the initial temperature both in Modelica and in 3D-ICE, avoiding the burden of keeping those values in sync.

IX. Simulation Examples

We have established that nowadays, for high-performance systems, the design of the cooling circuit must often be tailor-

made. We have also established that the said circuit needs assessing prior to facing potentially destructive stress, by joint simulation with the chip and the policies aboard. However, when designing the cooling circuit, sizing its components and setting up its controls, the above joint simulation is quite often not necessary owing to the band separation between on-silicon and circuit dynamics. It is apparently desirable that the same cooling circuit models can be used for any of the above activities, i.e., both with and without a connection to chip simulation codes like 3D-ICE.

Said otherwise, the workflow for designing a cooling system should start by concentrating on sizing heat sinks and hydraulic components, having in this initial phase the chips to cool just represented as prescribed thermal powers. Then, once the above and its controls are assessed, the obtained model needs to be connected to detailed chip simulators to investigate the behaviour of on-chip policies and perform the necessary fine tuning.

In this section we show a few examples to demonstrate how our approach – and as a consequence the presented library – do fully support such a workflow, that needs to concentrate sometimes on the chip alone, sometimes on the cooling alone, and sometimes on the compound of the two.

We would like to point out that the models here presented are custom cooling loops built with parts that have not yet been manufactured. Therefore, the reported examples demonstrate one of the major strong points of component-based simulation, i.e., the possibility to assess the performance of new solutions prior to their realisation. The reader interested in how our models are validated, and which accuracy can be achieved, can refer to the previous work [9]. In the research presented therein we modelled a specific water block, using the same approach and equations adopted in the following examples. In that case we could carry out a laboratory validation, and the average temperature error observed was 0.9°C, while the maximum error – temporary and during a transient – 5°C. Such a result is more than adequate for the intended purpose.

A. Example 1

We start with an all-Modelica example (i.e., no chip simulator is involved) to show the approach capability to address quite complex cooling circuit models including controls. The used model is shown in Figure 10: a centrifugal pump (a) feeds three modulating valves (b) to govern the flow rate through three waterblocks (c) connected each to one of three CPU-spreader compounds (d); three water-air exchangers (e) and a vented storage tank (f) close the circuit. Three PI regulators (g) control the three spreader temperatures, also governing the pump speed based on the water request. Boundary conditions are the external air temperature (h) and the power traces for the three CPUs (i). The DAE model corresponding to the scheme in Figure 10 has 1901 equations and 207 state variables.

As said, CPUs are modelled as prescribed powers, and only a spreader layer is represented: the relevant thermal characteristics and the TDP (91W) were taken from an Intel Core i5-6600K. Application traces have been collected using an Intel Core i5-6600K processor instrumented with a shunt

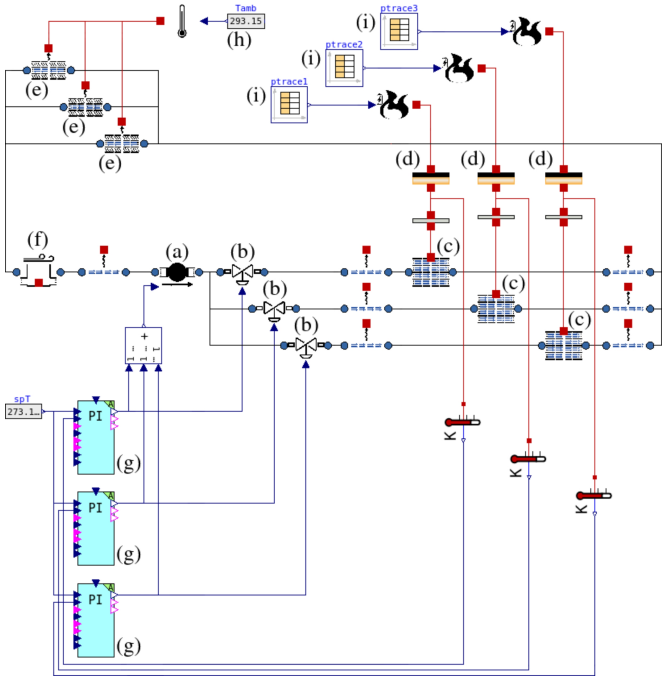


Fig. 10: Example 1 – Modelica diagram.

resistor between the 12V power supply and the connector on the motherboard dedicated to powering the CPU, thus excluding interference from the RAM and motherboard power consumption. Power traces were collected while running the Cloverleaf mini-application [33] from the UK Mini-App Consortium, which employs an explicit second order method for the resolution of compressible Euler equations, a representative application for the high-performance computing domain.

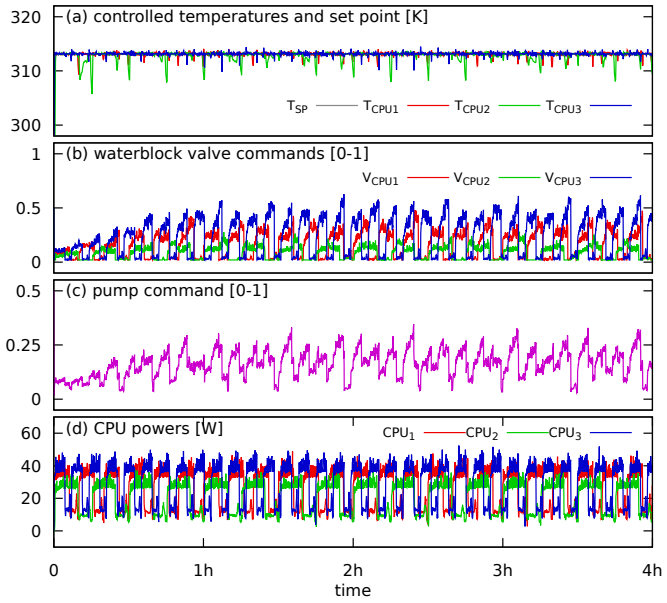


Fig. 11: Example 1 – simulation results.

Thermal simulations were instead performed on an Intel Core i9-12900K running Ubuntu 20.04.4, using OpenModelica

1.20.0∼dev-155-g379c110. The simulation results shown in Figure 11 demonstrate the opportunity of endowing modern cooling with controls, thanks to which good temperature management is achieved on the scale of some seconds; this cannot replace millisecond-scale control, typically via DVFS, but certainly helps. It is worth noticing that the 4-hour-long simulation took only 84 seconds, which is approximately 170× faster than real time.

B. Example 2

We now present an example including detailed chip simulation with 3D-ICE. The simulated system is composed of a waterblock heat exchanger fed by a chiller that provides a constant water inlet temperature of 24°C and a flow rate of 0.12 litres/minute. The waterblock is modelled in Modelica as a 1D cavity with a finned base of 3×3cm, in which turbulent water flow induces a convective heat exchange ruled by the Dittus-Bölter equation [34]. The waterblock Modelica model is connected, by means of the 3D-ICE 3.0 co-simulation interface, with the 3D-ICE model of a square flip chip with a side of 1.024cm. The chip floorplan features an array of 4x4 heat generation areas, each split in two, so as to model our Thermal Test Chip (TTC) platform [35].

Figure 12 shows two steady-state chip temperature maps with different heating patterns, both totaling a power of 60W. The upper temperature map comes from a uniform power distribution across the 16 heating elements. In this case, the maximum temperature of 66.7°C is observed in the central heating elements; temperatures in the bottom part of the chip map are slightly lower as this is where the water inlet is located.

The second simulation instead shows a hotspot *scenario*, where the heating elements are turned on in a chessboard pattern, and dissipate 7.5W each to still total 60W. As can be seen the steady-state temperatures are higher, reaching a maximum of 79.1°C, but also the temperature distribution changes: the highest temperatures are now at the corners, not at the center. The reason is that central heating elements can effectively spread heat laterally through the cold parts of the chip, thus reaching a lower temperature than the elements at the corners.

Both simulations were performed on a Core i9-12900KF server running Ubuntu 20.04.4 LTS, OpenModelica 1.20.0∼dev-155-g379c110 and 3D-ICE 3.1. The simulation time is 72s: 95% is spent in 3D-ICE for the high-resolution chip simulation, and only 5% by Modelica for the heat sink simulation, thanks to the different spatial discretisation granularities for the chip and the sink models [9].

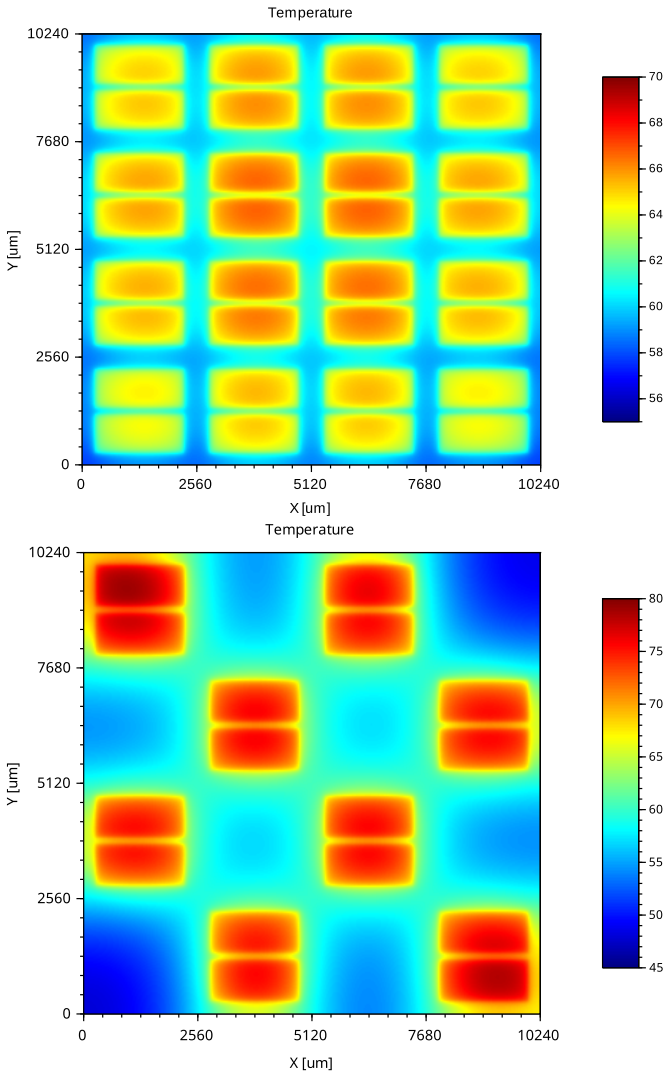


Fig. 12: Example 2 – simulation results: steady-state chip temperature maps with uniform (above) and chessboard (below) power distribution.

X. Conclusions and Future Work

We considered the problem of simulating modern MPSoC cooling circuits, that comprehend several heterogeneous physic domains, possibly together on-chip thermal policies. We showed that for an effective design of the entire system, the analyst must be able to concentrate sometimes on the chip, sometimes on the cooling system, and sometimes on the compound. We pointed out that the above requires a modelling framework capable of scaling the model complexity based on the study at hand, and that an enabling technology for this purpose is provided by the EBM paradigm.

In a recent paper we made the 3D-ICE simulator capable of cooperating with EBM tools, namely in the Modelica language, by exploiting the FMI standard for co-simulation. In this paper we complemented our contribution with a tutorial on how to write Modelica models for cooling circuit elements, illustrating the underlying principles and providing convenient examples. We also carried out laboratory validation activities, the description of which we did not repeat in this work.

To support our research, we have also initiated the development of a Modelica library of cooling component models. We hope that this library can become the first *nucleus* of an ecosystem of simulation models for modern cooling systems applied to MPSoCs and high-power ICs in general. We also hope – or better, we set as objective – that the availability of accurate, efficient, flexible and scalable simulators will in

turn enable the creation and assessment of innovative control strategies for optimal system operation. For this reason, as said, we are releasing our library (that is being continuously developed) as free software.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the financial support to perform this work by the ERC Consolidator Grant COMPUSAPIEN (Grant No. 725657) as well as by the H2020 project TEXTAROSSA (Grant No. 956831).

REFERENCES

[1] B. Zimmer et al., "A 0.11 pJ/Op, 0.32-128 TOPS, scalable multi-chip-module-based deep neural network accelerator with ground-reference signaling in 16nm," in *2019 Symposium on VLSI Circuits*, 2019, pp. C300–C301.

[2] P. Vivet et al., "IntAct: A 96-core processor with six chiplets 3D-stacked on an active interposer with distributed interconnects and integrated power management," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 1, pp. 79–97, 2021.

[3] K. Prabhu et al., "CHIMERA: A 0.92-TOPS, 2.2-TOPS/W Edge AI accelerator with 2-mbyte on-chip foundry resistive ram for efficient training and inference," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 4, pp. 1013–1026, 2022.

[4] V. Suresh et al., "Bonanza mine: an ultra-low-voltage energy-efficient bitcoin mining ASIC," in *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 354–356.

[5] F. Terraneo, A. Leva, and W. Fornaciari, "Event-Based Thermal Control for High Power Density Microprocessors," in *Harnessing Performance Variability in Embedded and High-performance Many/Multi-core Platforms: A Cross-layer Approach*, W. Fornaciari and D. Soudris, Eds. Springer International Publishing, 2019, pp. 107–127.

[6] A. Leva, F. Terraneo, I. Giacomello, and W. Fornaciari, "Event-based power/performance-aware thermal management for high-density microprocessors," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 2, pp. 535–550, 2017.

[7] G. Ali, L. Wofford, C. Turner, and Y. Chen, "Automating CPU dynamic thermal control for high performance computing," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 514–523.

[8] A. Seuret, A. Iranfar, M. Zapater, J. Thome, and D. Atienza, "Design of a two-phase gravity-driven micro-scale thermosyphon cooling system for high-performance computing data centers," in *2018 17th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2018, pp. 587–595.

[9] F. Terraneo, A. Leva, W. Fornaciari, M. Zapater, and D. Atienza, "3D-ICE 3.0: efficient nonlinear MPSoC thermal simulation with pluggable heat sink models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[10] Modelica home page, https://modelica.org/.

[11] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons, 2014.

[12] Functional Mock-up Interface (FMI) standard home page, https://fmi-standard.org/.

[13] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold *et al.*, "The functional mockup interface for tool independent exchange of simulation models," in *Proceedings of the 8th international Modelica conference*. Linköping University Press, 2011, pp. 105–114.

[14] P. Fritzson et al., "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development," *Modeling, Identification and Control*, vol. 41, no. 4, pp. 241–295, 2020.

[15] K. Arendt, M. Jradi, H. Shaker, and C. Veje, "Comparative analysis of white-, gray-and black-box models for thermal simulation of indoor environment: teaching building case study," in *Proc. 2018 Building Performance Modeling Conference and SimBuild*, Chicago, IL, USA, 2018, pp. 26–28.

[16] G. Agosta, E. Baldino, F. Casella, S. Cherubin, A. Leva, and F. Terraneo, "Towards a high-performance Modelica compiler," in *Proc. 13th International Modelica Conference*, Regensburg, Germany, 2019, pp. 313–320.

[17] S. Macchietto, *Dynamic model development: methods, theory and applications*. Elsevier, 2003.

[18] B. Kulakowski, J. Gardner, and J. Shearer, *Dynamic modeling and control of engineering systems*. Cambridge, United Kingdom: Cambridge University Press, 2007.

[19] F. Cellier and J. Greifeneder, *Continuous system modeling*. Springer Science & Business Media, 2013.

[20] S. L. Chaplot, R. Mittal, and N. Choudhury, *Thermodynamic properties of solids: Experiment and modeling*. John Wiley & Sons, 2010.

[21] R. C. Reid, J. M. Prausnitz, and B. E. Poling, "The properties of gases and liquids," 1987.

[22] F. Casella and C. Richter, "ExternalMedia: a library for easy re-use of external fluid property code in Modelica," in *Proc. 6th International Modelica Conference*, Bielefeld, germany, 2008, pp. 157–161.

[23] E. Lemmon, M. Huber, and M. Mclinden, "NIST standard reference database 23: reference fluid thermodynamic and transport properties – REFPROP, version 8.0," 2007.

[24] I. H. Bell, J. Wronski, S. Quoilin, and V. Lemort, "Pure and pseudo-pure fluid thermophysical property evaluation and the open-source thermophysical property library CoolProp," *Industrial & Engineering Chemistry Research*, vol. 53, no. 6, pp. 2498–2508, 2014.

[25] I. C. Turner, *Engineering applications of pneumatics and hydraulics*. Routledge, 2020.

[26] R. Karwa, *Heat and mass transfer*. Springer Nature, 2020.

[27] J. P. Holman, "Heat transfer," 2010.

[28] M. Oh and C. Pantelides, "A modelling and simulation language for combined lumped and distributed parameter systems," *Computers & Chemical Engineering*, vol. 20, no. 6-7, pp. 611–633, 1996.

[29] W. Minkowycz, E. Sparrow, G. Schneider, and R. Pletcher, *Handbook of numerical heat transfer*. New York, NY, USA: Wiley Interscience, 1988.

[30] R. Eymard, T. Gallouët, and R. Herbin, "Finite volume methods," *Handbook of numerical analysis*, vol. 7, pp. 713–1018, 2000.

[31] V. Lakshminarayanan and N. Sriraam, "A comparative study of heat transfer mechanisms from a vlsi integrated circuit die with and without a heat spreader." *Journal of Active & Passive Electronic Devices*, vol. 11, no. 1, 2016.

[32] S. S. Salvi and A. Jain, "A review of recent research on heat transfer in three-dimensional integrated circuits (3-d ics)," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 11, no. 5, pp. 802–821, 2021.

[33] A. Mallinson, D. A. Beckingsale, W. Gaudin, J. Herdman, J. Levesque, and S. A. Jarvis, "Cloverleaf: Preparing hydrodynamics codes for exascale," *The Cray User Group*, vol. 2013, 2013.

[34] R. Winterton, "Where did the Dittus and Boelter equation come from?" *International Journal of Heat and Mass Transfer*, vol. 41, no. 4-5, pp. 809–810, 1998.

[35] F. Terraneo, A. Leva, and W. Fornaciari, "An Open-Hardware Platform for MPSoC Thermal Modeling," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2019, pp. 184–196.

Federico Terraneo (M'21) has obtained the Master of Science Degree in Engineering of Computing Systems at the Politecnico di Milano, Italy with the thesis "Control based design of OS components", and the Research doctorate in Information Technology with the thesis "Thermal and energy management techniques for multi-core and many-core systems". He is currently Assistant Professor at Politecnico di Milano, Italy and his current research line focuses on thermal management for microprocessor-based computing systems and improving time determinism of distributed embedded systems. He has multidisciplinary research interests, including embedded systems and operating systems, with a lively interest in designinig hardware and software components applying a control-theoretical methodology. Since 2008 he is the original designer, main developer and maintainer of Miosix (http://miosix.org), an operating system kernel targeting microcontroller-based embedded systems.

Alberto Leva received the Laurea degree in Electronic Engineering (summa cum laude) in 1989 from the the Politecnico di Milano, Italy. In 1991 he joined the Dipartimeto di Elettronica, Informazione e Bioingegneria at the Politecnico di Milano, Italy, where he is at present Professor of Automatic Control. His main research interest concern methods and tools for the automatic tuning of industrial controllers and control structures, process modelling, simulation and control, particularly within the object-oriented paradigm, object-oriented modelling and control of energy systems with particular reference to large grids, and advanced tools and methods for control education. In recent years, he has been concentrating on control and control-based design of computing systems, addressing in a system- and control-theoretical manner problems like scheduling, resource allocation, time synchronisation in wireless sensor networks, thermal and power/performance management, performance-driven software adaptation, and service composition.

William Fornaciari is Associate Professor at Politecnico di Milano, Italy. He published 6 books and around 300 papers in int. journals and conferences, collecting 6 best paper awards and one certification of appreciation from IEEE. He holds three international patents on low power design and thermal management. He has been coordinator of both FP7 and H2020 EU-projects. In 2016 he won the HiPEAC Technology Transfer Award and in 2019 the Switch-To-Product competition. His research interests include embedded and cyber-physical systems, energy-aware design of sw and hw, run-time management of resources, High-Performance-Computing, design optimisation and thermal management of multi-many cores.

David Atienza (M'05-SM'13-F'16) is a Full Professor of electrical and computer engineering, and Head of the Embedded Systems Laboratory (ESL) at the Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland. He received his PhD in computer science and engineering from UCM, Spain, and IMEC, Belgium, in 2005. His research interests include system-level design methodologies for high-performance multi-processor system-on-chip (MPSoC) and low power Internet-of-Things (IoT) systems, including new 2-D/3-D thermal-aware design for MPSoCs and many-core servers, and edge AI architectures for wireless body sensor nodes and smart consumer devices. He is a co-author of more than 350 papers in peer-reviewed international journals and conferences, one book, and 12 patents in these fields. Dr. Atienza has received the ICCAD 2020 10-Year Retrospective Most Influential Paper Award, the DAC Under-40 Innovators Award in 2018, the IEEE TCCPS Mid-Career Award in 2018, an ERC Consolidator Grant in 2016, the IEEE CEDA Early Career Award in 2013, the ACM SIGDA Outstanding New Faculty Award in 2012, and a Faculty Award from Sun Labs at Oracle in 2011. He served as DATE 2015 Program Chair and DATE 2017 General Chair. He is an IEEE Fellow, an ACM Distinguished Member, and served as IEEE CEDA President (period 2018-2019) and Chair of EDAA (period 2022-2023).