



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Object-Oriented Modelling and Simulation of Computer Cooling Systems

TESI DI LAUREA MAGISTRALE IN
AUTOMATION AND CONTROL ENGINEERING - INGEGNERIA
DELL'AUTOMAZIONE

Author: **Marco Chioggi**

Student ID: 964545
Advisor: Prof. Alberto Leva
Co-advisors: Federico Terraneo
Academic Year: 2021-22

Abstract

All electronic components generate heat during operation. Traditionally, this heat was removed using air cooling systems, with heat sinks and fans. But nowadays, as IT equipment get faster and smaller, the amount of heat generated increases and it is more concentrated in compact volumes. In fact, some high-performance electronic components generate more heat than traditional cooling systems can dissipate and this brings to a research interest into new possible solutions, such as water usage and new innovative strategies. This new trend must be combined with the use of modelling and simulation tools looking for efficient and high-performance solutions. In this thesis work, an existent Modelica software library will be presented and improved in order to implement and adapt new models to the current research needs. This Modelica library contains several components that are useful to model and simulate different cooling systems, in addition, a template has been created and explained to make possible the interaction between Modelica and the 3D-ICE simulation platform, used to simulate and represent a virtual system in detail. To show the capability and evaluate the Modelica library a data centre rack system and its cooling system have been implemented and analyzed, all the results and comments are pointed out in this work. To conclude, an overview of possible future developments will be given.

Keywords: Object-Oriented Modelling, Computer Cooling, Thermal Modelling, Data Centre

Abstract in lingua italiana

Tutti i componenti elettronici generano calore durante il loro funzionamento. Fino ad oggi, questo calore veniva rimosso utilizzando sistemi di raffreddamento ad aria, con dissipatori e ventole. Oggigiorno, però, le attrezzature informatiche diventano sempre più piccole e performanti, generando molto più calore concentrato in volumi sempre più compatti. Alcune componenti ad alte prestazioni generano infatti più calore di quello che è possibile dissipare attraverso i tradizionali sistemi di raffreddamento, questo ha generato un notevole interesse da parte della ricerca verso nuove possibili soluzioni, come l'impiego di acqua e lo studio di strategie innovative. Per riuscire a trovare delle soluzioni ottime e performanti, lo sviluppo di questi nuovi sistemi deve essere combinato con l'impiego di software e piattaforme di modellazione e simulazione. Questa tesi si propone di analizzare e migliorare una libreria software già esistente e sviluppata in Modelica, aggiungendo e modificando alcune componenti in modo da espandere ed adattare la libreria allo stato dell'arte attuale. Questa libreria contiene quindi diverse componenti utili a modellare e simulare moderni sistemi di raffreddamento, è stato sviluppato inoltre un template in modo da rendere possibile l'utilizzo della piattaforma di simulazione 3D-ICE combinata con Modelica, questa può essere d'aiuto per ottenere simulazioni e rappresentazioni virtuali del sistema molto dettagliate. Per mostrare le funzionalità e le caratteristiche della libreria è stata modellata e simulata la struttura di un data centre, tutti i risultati e le osservazioni fatte sono presenti in questo lavoro. Per concludere, verrà offerta una panoramica dei possibili sviluppi futuri e migliorie che potrebbero essere implementate in futuro.

Parole chiave: Modellazione, Raffreddamento a liquido, Centro elaborazione dati

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
Introduction	1
1 State-of-Art	7
1.1 Modelica	7
1.2 3D-ICE	9
1.2.1 Inputs of 3D-ICE	10
1.2.2 Starting the simulation	11
1.2.3 Outputs of 3D-ICE	12
1.3 Computer cooling library	12
1.3.1 General description	12
1.3.2 Main packages	13
2 Computer Cooling	17
2.1 Solid components	17
2.1.1 Stream confinement	18
2.1.2 Uniform grid	20
2.1.3 Structured grid	22
2.2 Incompressible liquid components	23
2.2.1 Ducts	23
2.2.2 Waterblocks	25
2.3 Control blocks	25
2.3.1 Controllers	25
2.3.2 Actuation schemes	26

3	3D-ICE Implementation	27
3.1	Set-Up	27
3.1.1	Download Modelica library	27
3.1.2	Download 3D-ICE	27
3.1.3	3D-ICE templates	27
3.2	MyHeatSink Template files	28
3.2.1	Input files	28
3.2.2	Modelica cooling circuit	29
3.2.3	Results	30
3.2.4	Scripts	30
3.3	Example	30
3.3.1	Modelica scheme	31
3.3.2	3D-ICE parameters	32
3.3.3	Results	32
4	Example of a rack system	37
4.1	Liquid cooled rack system architecture	37
4.2	System set-up	39
4.2.1	Liquid cooling system	39
4.2.2	Rack system	41
4.2.3	Control system	44
4.3	Simulation results	46
4.4	Variations	49
5	Conclusions and future developments	53
	Bibliography	55
	List of Figures	59

Introduction

The Overheating problem

Heat is an inevitable byproduct of computer hardware operations, since the beginning of the electronic computing, overheating of electronic components was a primary problem to deal with in both private and industrial setting.

Considering the industrial application, nowadays, as the demands of remote data services keep increasing, both the workload of the data centre and its power consumption are rapidly rising, these systems accounts currently for over 1% of the world's electricity usage [1, 2]. For the ICT equipment, the operation temperature is an important factor that can greatly affect their stability and performance. Therefore some auxiliary systems, such as the cooling system and the power supply system, are used to ensure the stable operation of the ICT devices [3]. Energy efficiency becomes even more important for these systems, data from surveys suggests that the consumption due to the cooling and ventilation of the ICT equipment is between 30% and 55%, with an average of 40%, of the total energy consumption in a data centre [1, 4].

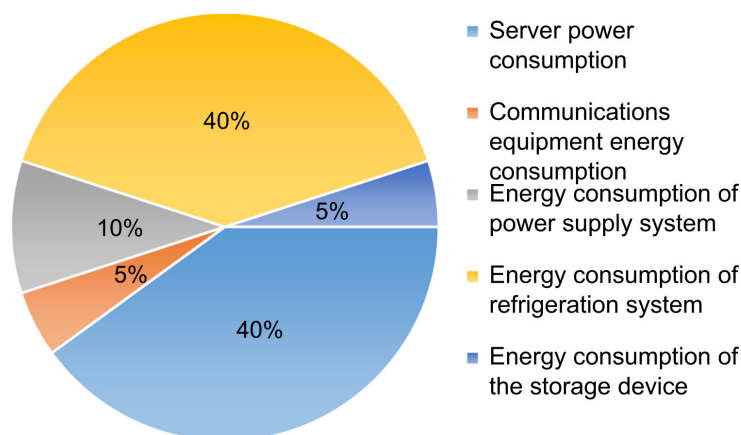


Figure 1: Energy consumption distribution of a data centre [5]

Taking into consideration the private setting, such as personal computers and mobile devices, there exist a trend toward more powerful and energy consuming microprocessor

combined with the higher packing density of the components.

Therefore it is clear that with these new trends and growth there is a need for research to be extended and improved. The design of new cooling strategies and the optimization of the process, for both private and industrial setting, plays a fundamental role [6].

Cooling strategies

Generally, a cooling system is made up of three different sub-systems [3]: heat generated during hardware operations of the IT equipment will be absorbed by a terminal cooling sub-system. Here heat is transferred to the primary heat sink by conduction and the main goal is to remove it away from the hardware itself. Different strategies can be applied, such as change the material and shape of the heat sink or using oils or gas to modify the thermal resistance. The second system transports heat from the heat sink to the refrigeration sub-system, this can be done in several ways, generally heat is transferred via one or multiple fluids. The last sub-system is the above-mentioned refrigeration one in which heat is dispersed into the outdoor environment.

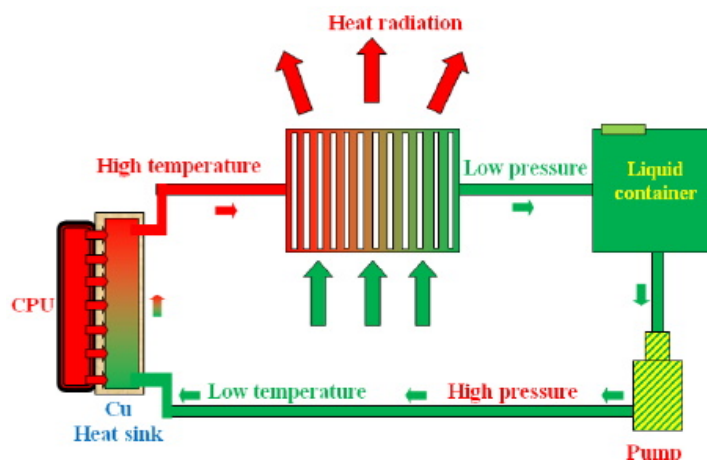


Figure 2: Example of a cooling system

Depending on the cooling principle, the last two sub-systems can be classified into three main technologies: air-cooling, liquid-cooling or hybrid-cooling [2]. Historically, forced- and natural-convection air-cooling has been the predominant method of cooling electronics. If additional cooling is necessary it is possible to implement a fan in order to increase the airflow over the heat sink to dissipate more heat, but this fills valuable space with the fan, fan mount and airflow entry and escape paths. This equipment becomes troublesome if the reduction of the space required becomes an important objective. Liquid-cooling counters almost every drawback of air-cooling, it can dissipate more heat with considerably less

flow volume and surface area, maintain a better temperature consistency, and do it with less local acoustic noise [7]. With a liquid-cooling technology there is the risk to damage the delicate electronics in the event of a leak, and generally the complexity of the required hardware to be controlled and managed added design and build costs [8].

In spite of these disadvantages, however, liquid-cooling strategies are essential in order to remove sufficient heat from the tomorrow's more powerful systems [8, 9].



(a) Air-cooling component



(b) Liquid-cooling component

Figure 3: Typical computer cooling components

Modelling and simulations

Modelling and simulations enable designers to test whether design specifications are met by using virtual rather than physical experiments. In addition, experiments on management of large scale computing systems are difficult or not possible to conduct in real environments. The use of virtual prototypes significantly shortens the design cycle and is particularly cost-effective. It further provides the designer with immediate feedback on design decisions which, in turn, promises a more comprehensive exploration of possible alternatives and better performing final results [10].

Simulation is particularly important for the design of multi-disciplinary systems in which components of different physical systems (mechanical, electrical, thermal, electromagnetic, etc.) with different time-scaled dynamics are tightly coupled to achieve optimal results [11, 12]. The modelling procedure usually leads to a set of differential algebraic equations, simulation is then conducted to numerically solve the mathematical equations in order to calculate the system dynamics [12].

Many tools have been developed in academia and industry to perform computer modelling and simulation of cooling systems, most of these are based on imperative programming languages such as FORTRAN(e.g., TRNSYS[13]) and C/C++(e.g., Energy-Plus[14]).

This approach has several limitations: the non-linear equations are usually manipulated to be solved iteratively, and the differential equations are discretized to numerically approximate the state variables. Therefore, the above-mentioned conventional simulation tools expose several disadvantages in terms of their modelling and simulation performance. In addition, these tools are hardly suitable on various user's needs and some traditional languages are causal (e.g., MATLAB/Simulink) and do not support acausal modelling [11, 12].

Modelica [15], on the other hand, is a modelling language governed by an open standard, that can effectively address these limitations. First, is an equation-based and object-oriented language that supports both causal and acausal modelling and, secondly, allows developers to build multi-disciplinary system models featuring both thermo-fluid system models (typically acausal) and realistic controls (typically causal) using a graphical and hierarchical approach. These features allow models to be constructed with classical physical equations (i.e., not constrained to only input/output formulations) and may contain continuous, discrete, and hybrid differential equations, aiding flexibility for use cases and significantly reducing model development time [16]. Simulation code is generated automatically, and in contrast to many traditional tools, the model equations are separated from the simulation code. This improves the developer's ability to successfully simulate complex system models, with both fast and slow dynamics, which are common in computer cooling applications.

Lastly, the Modelica community has rich open-source libraries that span multiple domains, which enable users to construct their case study models from similar systems and share resources among external groups [12].

AIM of the work

The aim of this thesis work is to extend and improve an existent Modelica software library containing hydraulic and thermal components [17]. The goal of this library is to provide a complete set of models that can be used to easily design computer cooling systems. The library has an user-friendly interface with the simulation tool 3D-ICE [18] used to simulate and test different cooling structures in a virtual environment.

In the first part of the thesis some basic notions about Modelica [15] and 3D-ICE [18] will be introduced in order to simplify the following description of the models and scripts. Then the existent library will be presented [17] and a brief overview over what contains and its state of art will be given.

The second part of this thesis is dedicated to the description of the new models added to

the library, mainly those that are typical of a liquid-cooling system, and how can be used and improved.

A third part will focus on the interface between Modelica and 3D-ICE and on the template used to integrate the two software [19]. Some scripts will be introduced in order to plot the simulation results and analyze the cooling strategies.

In order to validate the models and evaluate the simulating performance of the library, an example of a data centre rack cooling system will be shown [20, 21]. In addition, a basic control strategy is applied to prove the capability of the library in the target to reduce energy consumptions and optimize the cooling process.

Lastly, this thesis will discuss future development of the library and improvements that can be made.

1 | State-of-Art

This first chapter is dedicated to the introduction of Modelica and 3D-ICE platform. The aim is to show the very basic software notions in order to better understand the following contents of the work. In addition, the starting Modelica library will be introduced [17], its structure and the existent models will be briefly analyzed.

1.1. Modelica

Modelica is an object-oriented, equation-based and declarative modelling language for complex systems modelling [22]. Its multi-domain capabilities allow designers to capture and integrate the dynamic behaviour of any physical area. Every modelled components can then be combined into sub-systems, systems or even architectures. Models are represented both graphically and in code and can be easily adapted to fit specific needs, moreover, as mentioned in the introduction, each model is described in terms of constitutive set of equations in which there is no explicit specification of system inputs and outputs. The physical equations of each component are combined with energy conservation ones to determine the overall set of equations to solve, their resolution and manipulation is left to the simulation engine [23].

The free Modelica language is developed by the non-profit Modelica Association [22] for all operative systems, which also provides open access to the Modelica Standard Library [24] that contains lots of basic models for a wide range of applications. In addition, there are math functions, utilities and examples for several domains.

In this section the basic functionalities and notions will be described [25, 26].

Packages

A **package** is simply a container, a directory, used to organize models, functions, constants, and other allowed contents. The package name is prefixed to all definitions within the package itself using standard dot notation. Packages can contain sub-packages and can store its models inside a proper directory, this can be useful in order to better arrange files.

In order to define a package the syntax `package <package_name>` and `end <package_name>` must be used and if a package or a model is nested in another package the clause `within <root_package_name>` must be added.

In each package folder there are different Modelica files marked by the extension `.mo` that represent the models inside the package, in addition, there is a file with the extension `.order` used by Modelica to establish the order of the Models and sub-packages inside the package.

Models

A `model` is a behavioral description, it can be similarly juxtapose to class in object-oriented programming languages. As the `package` syntax, the keyword `model` indicates the start of the model definition and the key word `end` closes it. Models are divided into two sections: the first one is used to declare variables and to list all the inheritance and extensions or declarations of other models. the keyword `equation` denotes the second section in which the physical (algebraic and/or differential) equations that represent the component are listed.

An important clarification about models is the existence of the so called `partial model`, this is a partial class since it does not contain enough equations to completely specify its physical behavior, and is therefore prefixed by the keyword `partial`. These models work as parent class, are used to declare basic parameters, variables and equations of bigger models that will inherit from them. Partial classes are usually known as abstract classes in other object-oriented languages.

Variables

Any declared quantity has a specific variability, in Modelica there exist three kinds of variables:

First there are the `parameters` that represent quantities which remain constant during the simulation but may have different values from one simulation to another, users can change them before starting the simulation.

Then there are the `constants` that are quantities wick are unlikely to change, they must have an expression for the value of that constant.

Lastly there are the `variables` which quantities change during the simulation, these variables need to be declared specyfing also the type: `Real`, `Integer`, `Boolean`, `String`, etc.

Each declared quantity has a set of **attributes** that can be associated either with the type of the quantity. Some examples are: **start** to put a reasonable initial guess, **fixed** in order to fix the value to the initial guess, **min/max**, **unit** to set the unit for a type, etc.

Connectors

A **connector** allows to match up the appropriate variables from connectors of different components, more precisely, connectors are small models designed to make different components exchange information. When two or more connectors are connected, some equations are generated to correctly match the variables of each component.

Modelica supports equation-based acausal connections, which means that the direction of data flow in the connectors do not need to be specified. Additionally, causal connections can be established by connecting a connector with an input attribute to one declared as output. Two types of coupling can be established by connectors depending on whether the variables are declared **non-flow** (default), or declared using the prefix **flow**: for **non-flow** (also called **across**) an equation is generated which sets the matching components equal to each other, for **flow** variables the equation generated sums the matching components to zero. To define a **connector** the syntax is **connect**(port1,port2).

Functions and algorithms

During the development of models, there are cases where a procedural or algorithmic approach is necessary. To address this need, Modelica includes support for algorithmic functions. A **function** has only quantities labelled as **input** or **output** and it is not connected to other components. When a function is called using positional argument association, the number and the types of actual arguments and formal parameters must be the same.

To define a **function** the syntax is **function** <function_name> and **end** <function_name>, within an algorithm statement the assignment operator " := " must be used.

1.2. 3D-ICE

3D-ICE, or “3D Interlayer Cooling Emulator”, is a Linux based simulation platform written in C used to simulate the transient thermal behavior of 3D IC structures [18, 27].

This open source thermal simulator is based on the compact modelling of heat transfer by conduction in solids and on a new compact modelling methodology, called the "Compact Transient Thermal Modelling" (CTTM) [28]. This model is based on the identification

of the equivalent electrical representation of convective heat transport in fluid flows, as a voltage-controlled current source. This leads to a subdivision of heat transferred by convection in microchannels, users can size these partition depending upon the accuracy and speed needs.

This simulator is ideal for situations where a quick estimate of chip temperatures is required, when the aim is to iterate between various floorplanning and operating strategies in order to optimize the performance and thermal safety/reliability of the final system, as in the case of this work.

In this part the general basic concepts will be described [27]. The detailed files used for the computer cooling work are specified later.

1.2.1. Inputs of 3D-ICE

In order to provide to 3D-ICE all the information needed to emulate a heatsink-processor model, several input files must be defined, these data are stored in the heatsink model folder.

Build FMI

This file (*.mos) is the Functional Mock-up Interface (FMI), it defines a communication protocol to correctly and easily couple the dynamic models of 3D-ICE with other modelling tools, such as Modelica.

Floorplan

This file (*.flp) describes the thermal model architecture of a 3D IC. It contains all the information about the location and the size of each functional block in the IC (cores, caches, memories, etc...) and the corresponding heat dissipation traces, as a function of time. Each die must have a corresponding floorplan file stating the position, based on a cartesian grid (in μm), the dimensions of the heat spreader (in μm) and the dissipation profile (or heat sources) for the simulation. Every block, called floorplan element, has a unique identifier name and represents an area inside the die, laid out in the source layer.

Stack

The stack description file (*.stk) is the project file created by the user to describe the 3D IC thermal problem that will be solved. It contains information about the structure, material properties of the 3D Stack, the description of the various heat sinks in the system,

the discretization parameters, analysis parameters, and finally, commands to 3D-ICE for printing out the desired outputs from the simulation.

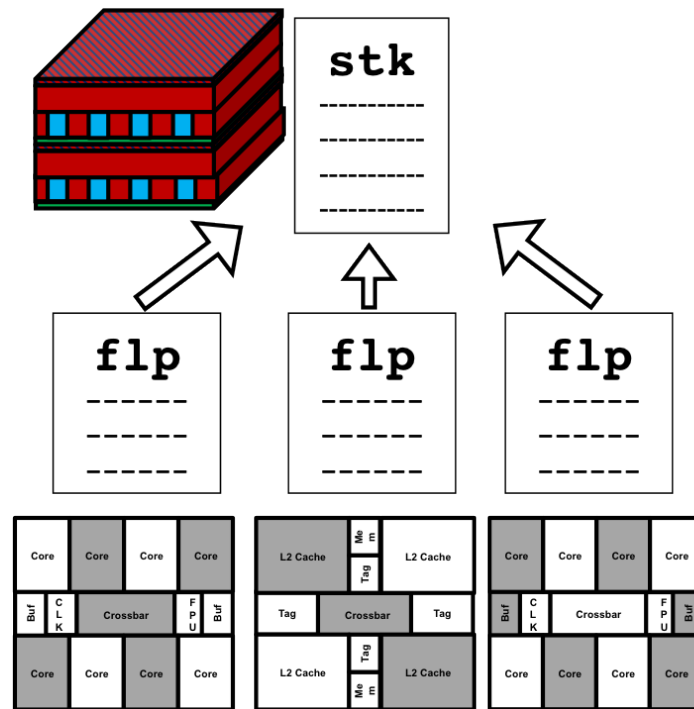


Figure 1.1: Floorplant and stack files [27]

Makefile

Makefile is used in order to couple the Modelica (*.mo) files with the FMI. It lists all the Modelica packages and models needed for the simulation.

Simulate

The `simulate.sh` file is launched by the terminal to start the simulation.

1.2.2. Starting the simulation

Once the input files are completed the user can start the simulation by simply opening a terminal in the heatsink folder and running the command:

```
./simulate.sh
```

1.2.3. Outputs of 3D-ICE

At the end of the simulation, additional files will have been created according to the instruction put inside the stack file. A new folder can be created in order to store the results of the simulation, such as processor temperature collected in a precise position or a general temperature map.

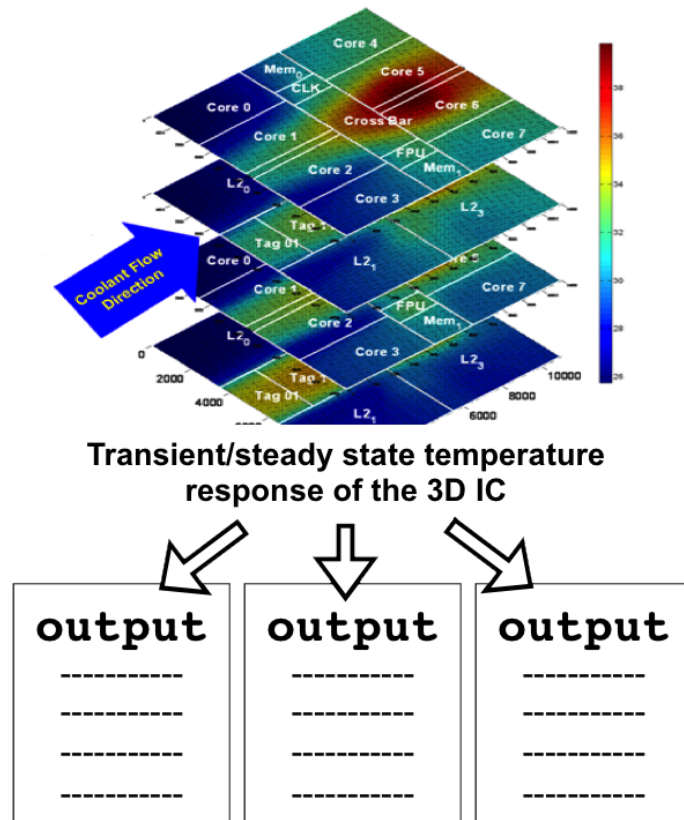


Figure 1.2: 3D-ICE outputs [27]

1.3. Computer cooling library

In this section the structure of the starting Modelica library will be presented, the main packages are listed below with a brief description of what they contain and the purpose of the models inside. A detailed description of some models will be presented in the next chapter. For a complete overview of the whole library consult [17].

1.3.1. General description

The Computer Cooling package is built on Modelica Standard library 3.5.3. It is subdivided into several sub-packages in order to better group models for different purposes and

physical domain, such as hydraulic components, sensors, heat transfer components, moist air components, and so on. The library class hierarchy is shown in the figure 1.3, as can be seen models generally inherit from base classes that can be reusable, that makes the use of the library more intuitive and approachable.

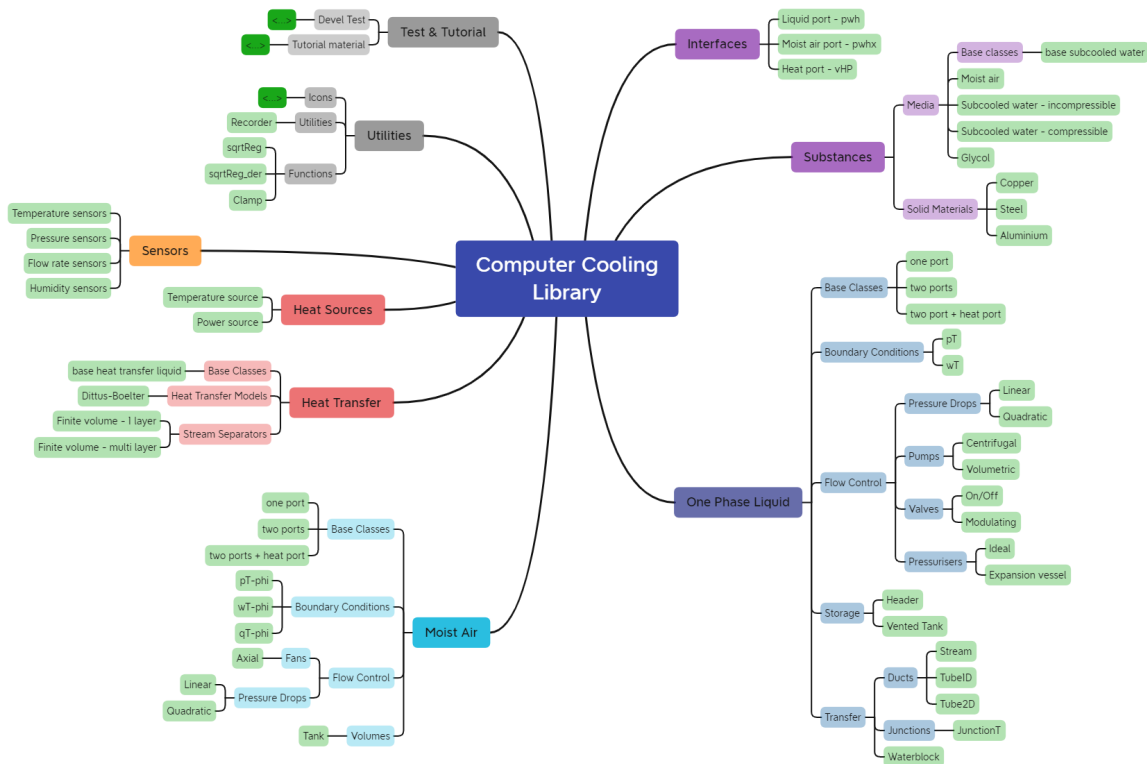


Figure 1.3: Diagram of the Starting Computer Cooling library [17]

1.3.2. Main packages

Icons

The `Icons` package contains a set of partial models that are useful for the navigation and the clarity of the whole library. The partial models inside this package are basically graphic marks useful to represent several elements, models and other tools within the library.

Utilities

The `Utilities` package contains only the `Recorder`, it is a tool that can be used to save the results of a simulation in a file. It is possible to modify the data time-step, the title

of the output file, to change the number of data to save and to add header in order to name the data collected.

Functions

Inside this package there is a collection of **Functions** that can be useful to compute some parameter values during the development of other models. A function is also useful to better organize the parameter definition and the equation section within models.

Interfaces

The **Interfaces** package contains the connectors used by the other models of the library. The elements inside allow the connection between models, indeed, they provide the "coordinates" of the fluid state that is entering or exiting the component the port will be part of. The parameters can be for example the fluid pressure p , the mass flow rate w , the enthalpy h , the mass fraction but also temperatures and power flows.

Solid materials

SolidMaterials includes a set of records that can be useful to model the behavior of several solid materials. These elements defines the value of basic attributes of materials such as density ρ , specific heat capacity c and thermal conductivity λ . Each record extends the base class that is common for all the solid materials.

Media

The **Media** package contains, as before, the base class and several models that represent the behaviour of different media such as incompressible water, moist air, glycol, etc... This package includes also a constants package used to list all the standard constants for convenience.

One phase liquid components

The **OnePhaseLiquidComponents** (OPLC) package includes all the models developed to work with a medium that does not change from liquid state. Generally speaking all the models inside this package are structured similarly; in the first part of the variables definition it is specified if the model has one or two hydraulic ports and a heat port, then the medium and the possible presence of a solid material are specified. After the declaration of all other variables the equations are listed, starting from those that

represent the current status of the medium, followed by dynamics equations and energy computations at the ports. This package contains boundary conditions, pumps, valves, tanks, ducts, and so on. The various components are not examined individually, with the exception of some models that have been improved in this work.

It is worth noticing that all the models inside this package are now developed to work only with incompressible liquid and so, within this work, it has been renamed into `IncompressibleLiquidComponents`.

Moist air

The `MoistAirComponents` package (MAC) groups all components designed to work with a moist air flow that accounts also of humidity variations. The structure of this package is similar to the previous one and it contains fans, pressure drops, tanks, ducts, etc...

Heat transfer

The `HeatTransfer` package includes components that model heat flow dynamics, this contains boundary conditions, heat sources and heat transfer models. The heat sources are defined positive if flows out of the component.

Sensors

The purpose of the Computer Cooling library is to develop a cooling circuit that can be later implemented and tested inside the 3D-ICE environment. Without a specific tool is impossible to collect data from the cooling circuit during or after the simulation. Hence, inside the package `Sensors` are listed several models representing sensors in order to capture data and collect them in the `Recorder` utility.

2 | Computer Cooling

In this chapter the improvements to the pre-existent Modelica library will be presented in detail. As discussed before these upgrades have been developed in order to better model and simulate modern computer cooling strategies, with a focus on liquid systems. All the new models are grouped and presented according to the sub-package of which they belong and their order within the library.

All the developed components, as mentioned before, are defined in a precise way: with the instantiation of required ports or the extension of a base class, declaration of the eventual medium and solid material, followed by parameters and variables initialization. Then, the equations that indicate the current status of the medium are stated before, followed by dynamic equations and energy considerations at the ports.

2.1. Solid components

The `SolidComponents` package holds all the models that represent a solid element in which the heat flows between its layers or lumps. This includes stream confinement for tubes and planar walls that represent heat spreaders and chip. In all the models contained in this package is possible to replace easily the material of the component, by choosing one from the `SolidMaterials` package.

To correctly model the heat flows between solid components is necessary to compute the thermal conductance \mathbf{G} and the heat capacity \mathbf{C} of each lump, these values are computed according to the following relationships:

$$\mathbf{G} = \lambda \cdot \frac{\mathbf{A}}{\mathbf{Dist}}, \quad (2.1)$$

$$\mathbf{C} = \mathbf{c} \cdot \rho \cdot \mathbf{Vol}. \quad (2.2)$$

Where \mathbf{A} is the heat transfer surface of a single volume lump, \mathbf{Dist} is the distance between the centre of two consecutive lumps and \mathbf{Vol} is the volume of a single lump.

2.1.1. Stream confinement

The `StreamConfinement` package holds the models that represent the stream confinement within tubes. First, a base class model is developed and then two different geometries are considered: planar and cylindrical walls. These two models describe a pipe with a single or multiple cross layers between the inner and outer heat interfaces, the pipes are also divided in finite volume lumps along the length-wise direction. An important detail is that the equations in these models describe the heat exchange between the inner and the outer heat boundaries only, the "horizontal" heat flows between the finite volume lumps are, instead, not modeled in these components.

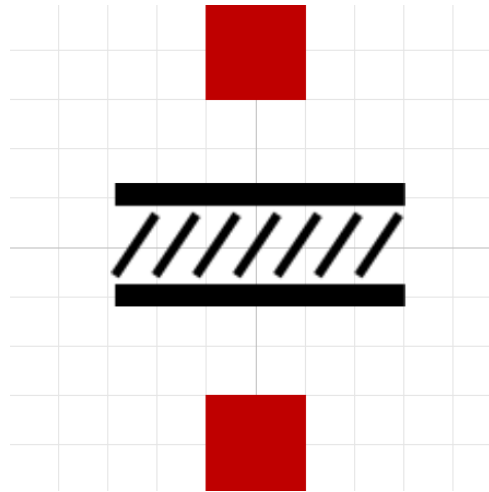


Figure 2.1: Stream confinement component (icon view)

Base class

In the `Base_StreamConfinement` partial model two vector heat ports are instantiated representing the inner interface (toward the fluid inside) and the outer interface (toward the environment). A solid material (copper by default) is also instantiated as a replaceable record to account for the thermal characteristics of the material. Then the geometrical parameters are defined, together with the temperature matrix and its initial values. It is also possible to set the number of volume lumps and the number of layers.

Planar wall

This model represents a planar stream confinement, it inherits the parameters and ports from the base class but, in order to compute the cross area, the width of the component has to be defined. Accordingly to what said before the equations in this model describe, for each finite element i , the heat exchange between the inner and the outer interfaces,

the complete set of equations is the following:

$$Q_{int|i} = \frac{1}{2}G(T_{int|i} - T_{1,i}), \quad (2.3)$$

$$CT_{i,j} = G(T_{i,j-1} - T_{i,j}) - G(T_{i,j} - T_{i,j+1}), \quad (2.4)$$

$$Q_{ext|i} = \frac{1}{2}G(T_{ext|i} - T_{m,i}). \quad (2.5)$$

Cylindrical wall

The Cylindrical wall model is similar to the previous one, with the only difference on the geometry of the element. In order to compute the heat capacity and the thermal conductance is sufficient to know the internal diameter and the total thickness of the pipe. These parameters are placed inside a vector because their values change depending on the layer that we are considering. Indeed the set of equations used is the same of the planar case.

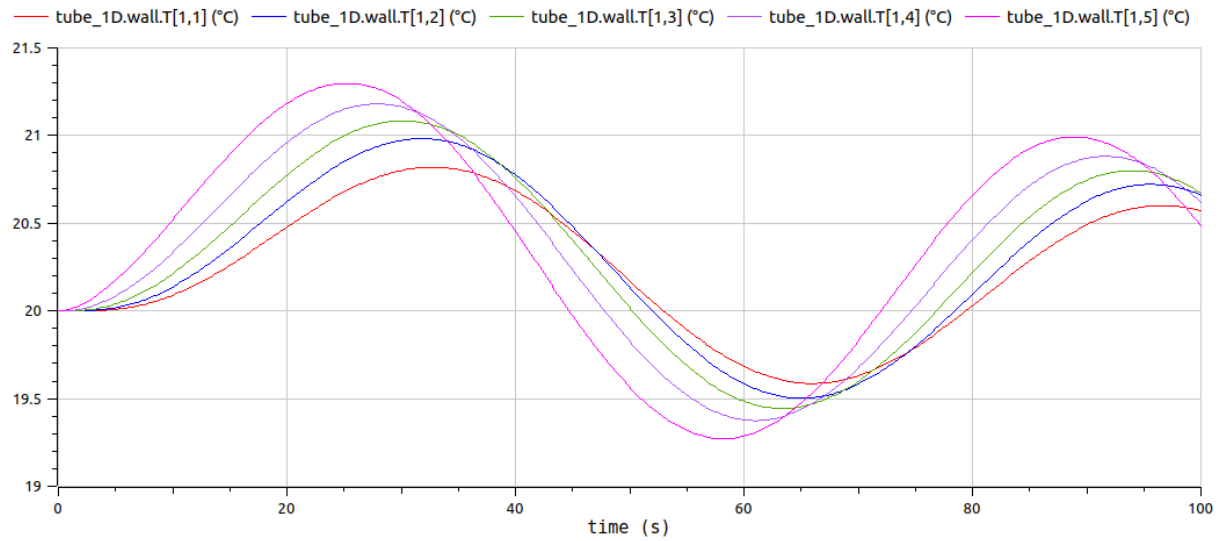


Figure 2.2: Heat propagation through a CylindricalWall_FiniteVolume model

2.1.2. Uniform grid

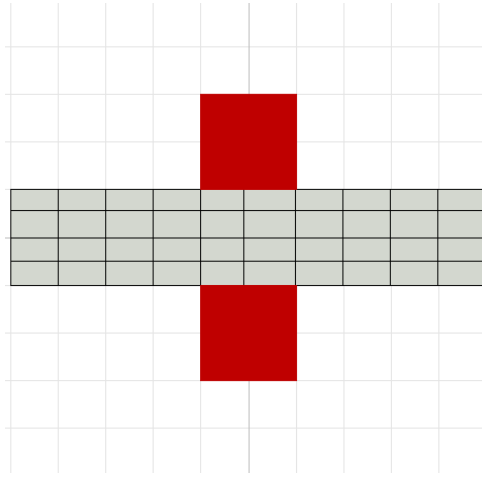


Figure 2.3: PlanarWall_MultiLayer_UniformGrid component (icon view)

The `PlanarWall_MultiLayer_UniformGrid` model is used to simulate solid components such as spreaders or chip made up of a specific solid material. In order to study the temperature distributions in these solids, a finite-volume approach must be used, in fact the solid is decomposed into parallelepiped elements. Two heat ports are instantiated representing the inner and the outer interfaces, then an user can set the total length and the number of lumps along all the three directions (width, depth and thickness). The model instantiates the temperature value of a single volume lump inside a temperature 3-dimensional matrix. The uniform grid case has the peculiarity that all the elements have the same size and so it is sufficient to compute only three thermal conductances values (one for each direction) and one heat capacity, because the volume of a single lump is the same for all the finite elements.

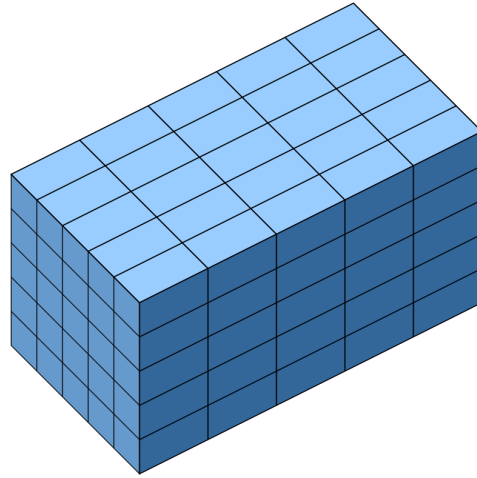


Figure 2.4: Uniform grid solid component

To correctly model and quickly simulate the heat exchange within the solid component is necessary to consider the temperature 3D matrix and write the exchange equations without considering each finite volume as a single component, this means that is not necessary to take into account the temperature at the faces of the finite volumes but, to do so, we need to handle separately all possible cases where one or more volume faces exchange heat with others volumes. This brings to a discretization of volume lumps into different groups, as shown in the figure 2.5. Each group has its exchange equation and its **for** loop.

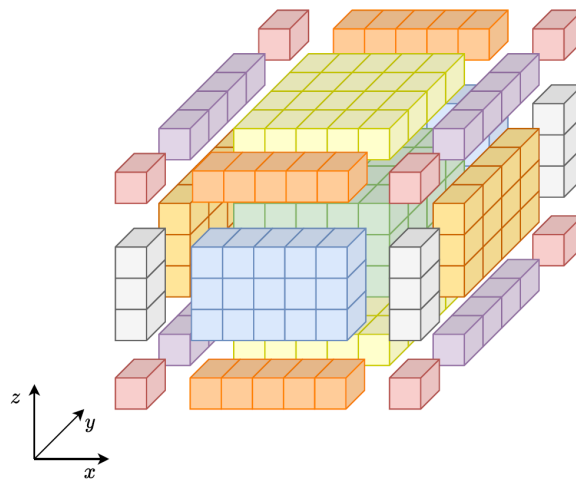


Figure 2.5: Volume decomposition via uniform grid

As an example, below are shown the **for** loops required to simulate the heat exchange between the lumps in the "central" part of the solid component (green elements in

figure 2.5). Denoting by w, d, t the number of elements along the three directions and by T the 3-dimensional matrix of volume temperatures, the equations read:

```

for i in 2:w-1 loop
  for j in 2:d-1 loop
    for k in 2:t-1 loop
      C*der(T[i,j,k] = G_w*(T[i-1,j,k]-2*T[i,j,k]+T[i+1,j,k])
            +G_d*(T[i,j-1,k]-2*T[i,j,k]+T[i,j+1,k])
            +G_t*(T[i,j,k-1]-2*T[i,j,k]+T[i,j,k+1]));
    end for;
  end for;
end for;

```

2.1.3. Structured grid

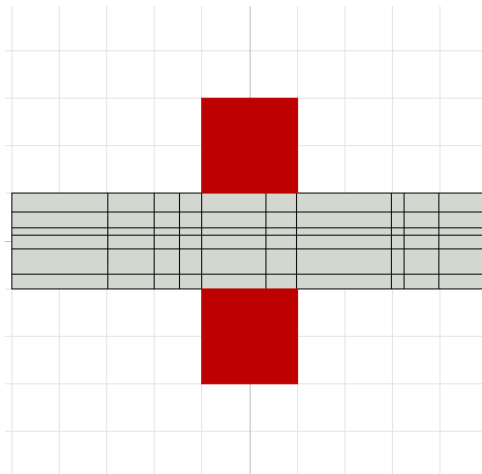


Figure 2.6: PlanarWall_MultiLayer_StructuredGrid component (icon view)

The PlanarWall_MultiLayer_StructuredGrid model is similar to the previous one with the exception that the lumps dimensions along each direction may be different from one lump to another. This brings to a more complex system and parameter definition, in fact, for this model is necessary to collect an array of lump dimensions along all the directions to make the computation of the single volumes possible. A temperature 3D array is instantiated as in the previous case but now the areas and distances must be computed for each finite volume and along each direction respectively. The thermal conductance and the heat capacity values are stored inside 3D arrays so as to compute the heat transfer in all directions. The loops used to model and simulate the heat exchange are the same as before, the solid must be divided as in the previous case, but now the thermal conductance

G and the heat capacity C within the equations depend on which finite volume we are considering.

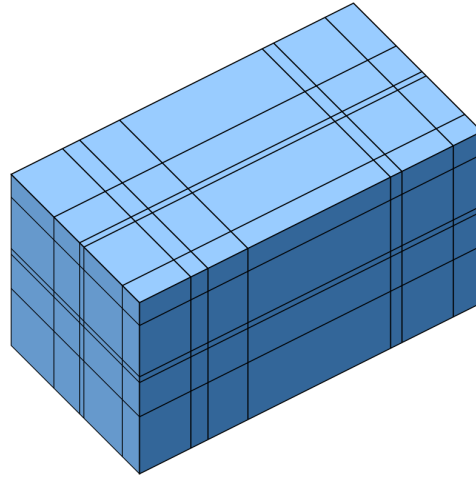


Figure 2.7: Structured grid solid component

2.2. Incompressible liquid components

The `IncompressibleLiquidComponents` package contains all the models that are designed to work with an incompressible and single phase medium which is instantiated via a replaceable model, so as to make easy for users to replace it with other media . Within this work the `IncompressibleLiquidComponents` package has been improved; the `ducts` sub-package has undergone a refactoring and a new model for the waterblock has been implemented. The following section will examine the various components.

2.2.1. Ducts

This sub-package contains components that are used to represent and simulate a liquid stream flowing inside a tube, these models are structurally the same of those developed in the original library [17]. During the refactoring only small changes has been applied and now an user can choose between ducts with a generic and cylidrical section. The subdivision into single layer and multi layer stream confinement is no more necessary because all these models can treat both cases, as specified before. Users are now free to choose the number of volume lumps and the number of layers, the only difference between the two models is on the geometrical point of view.

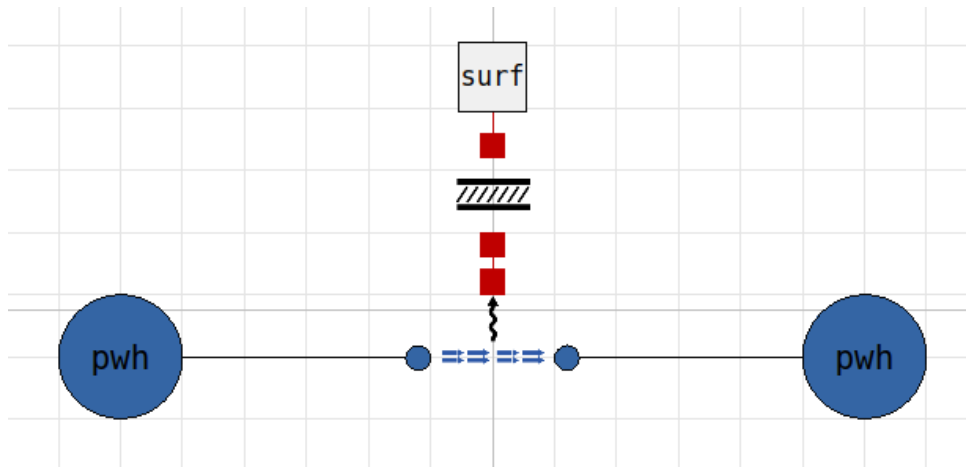


Figure 2.8: Tube_1D_Cylindrical model (diagram view)

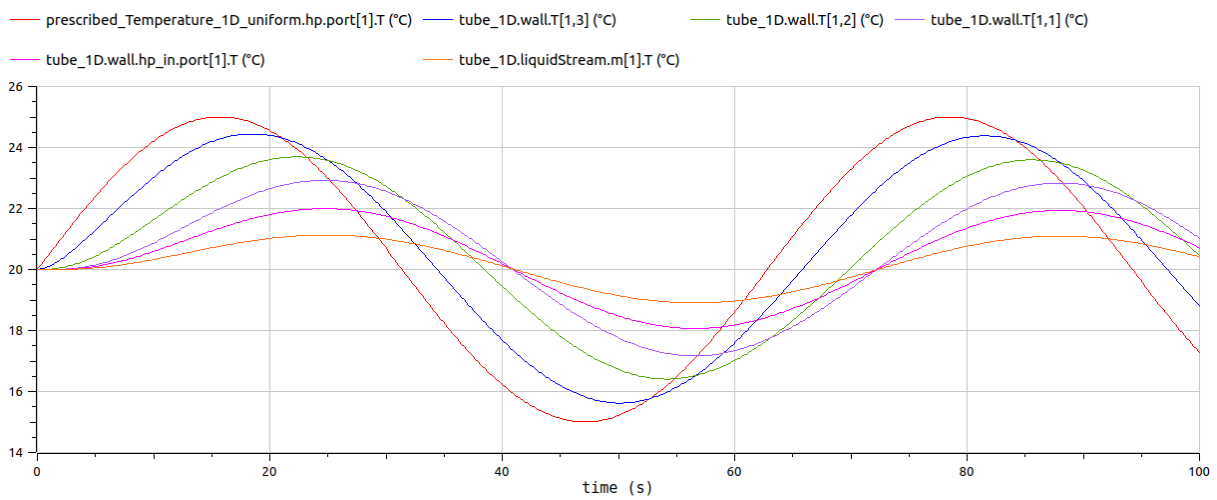


Figure 2.9: Heat propagation from external heat port to fluid

2.2.2. Waterblocks

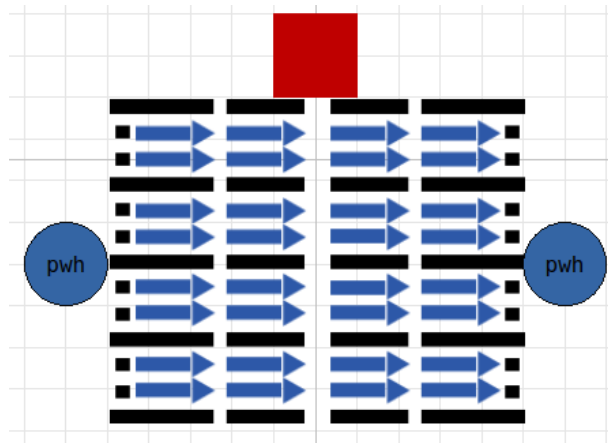


Figure 2.10: `WaterBlock_GenericSection_uniform_w` component (icon view)

The `waterblock` model has been developed specifically for liquid cooling systems, this component is used to represent the heat exchange from the processor (the source) into the liquid that flows through the channels within the waterblock, these channels are designed so as to increase the heat exchange surface. This Modelica model instantiates an array of `LiquidStream_FiniteVolume` components, these streams are the aforementioned channels and form a grid over the processor. the mass flow that comes from the input `pwh` interface is divided equally between each channel. The `waterblock` component has also a matrix heat port connected to each channel via code to enable the heat exchange between the processor and the liquid. Users can define the geometrical values and can set the number of lumps and channels.

2.3. Control blocks

To better understand the example that will be developed later within this work, it is necessary to do an overview on the `ControlBlocks` package. It contains analogical and digital controllers and some actuation schemes. These elements will be presented in this section.

2.3.1. Controllers

The controllers implemented in this library are subdivided into analogical and digital. The purpose and the explanation of the controller schemes will not be presented within this work because the used structures are already well covered in literature.

2.3.2. Actuation schemes

the Actuation schemes represent, at the control synthesis level, how the controller drives the actuators. The Actuation schemes developed for this library are listed below.

Split range

The purpose of the split range scheme is to consider two actuator as a single one, splitting their action in a different range of the control variable. As can be seen inside this model a deadzone is introduced in order to avoid a continuously switching of the two actuators. The two output control actions are enabled when the input signal is above or below the deadzone respectively.

Daisy chain

The purpose of this scheme is to activate several actuators in sequence, an actuator is enabled when the previous one has reached its maximum limit. The basic idea behind this scheme is to start with the most efficient actuator and then switch to the less efficient only if the previous one is not enough.

Counter Act

This actuation scheme has two outputs. If we consider an input control action with values in the sector $[0,1]$, the first output value is equal to the input, while the second one is the complementary value, i.e. the necessary value to reach 1.

Rescaling inputs

The purpose of this scheme is to rescale the input value, an user can choose the boundary values for 1 and 0, the maximum and the minimum respectively. Then this model rescale the input value $[0,1]$ within these boundaries.

3 | 3D-ICE Implementation

In this chapter a complete guide on how to set up a simulation will be given, starting from the download of the Modelica Computer Cooling library. In addition, an example will be treated to show how to plot and analyze the results using Scilab scripts, located within the 3D-ICE template in the Computer Cooling repository.

3.1. Set-Up

3.1.1. Download Modelica library

Once a Modelica release is installed, to download the library object of this work, navigate to the folder where the repositories will be saved and, from there, run the instruction:

```
git clone https://github.com/looms-polimi/computer_cooling.git
```

The Computer Cooling library is developed to work within Modelica 3.2.3 environment. Hence, it is necessary to install a compatible standard library to correctly use the Computer Cooling library.

3.1.2. Download 3D-ICE

Within the Computer Cooling folder, you will find a `readme` file that lists the instructions to install the dependencies and to download the 3D-ICE software which must be installed at the same directory level of the `computer_cooling` repository.

3.1.3. 3D-ICE templates

3D-ICE supports a pluggable heat sink interface that allow it to perform co-simulations with a separate heat sink model. It is possible for users to create their own heat sink model and then use 3D-ICE to simulate its behaviour, as in our case. To do so the Computer Cooling library has a 3D-ICE template folder, that must be copied outside the computer cooling git repository, at the same directory level of the `computer_cooling` and 3D-ICE

folders. The procedure to correctly copy the template is shown in the same `readme` file.

To correctly run a simulation, the folders structure must be as the following one:




 3d-ice	18 items
 computer_cooling	5 items
 MyHeatSink	8 items

Figure 3.1: Directories scheme

3.2. MyHeatSink Template files

In this section the files inside `MyHeatSink` folder will be analyzed, starting from the 3D-ICE input files already discussed in chapter 1.







 results
 scripts
 buildfmi.mos
 example.flp
 example.stk
 Makefile
 simulate.sh
 T05_3DICE_Integration.mo

Figure 3.2: Contents of MyHeatSink directory

3.2.1. Input files

These files have been already presented in chapter 1, they are defined as usual with some slightly differences in order to adapt the Modelica library to the 3D-ICE software. The

FMI, floorplan, makefile and stack files also load the Modelica library package in order to give access to 3D-ICE to the heatsink Modelica components.

3.2.2. Modelica cooling circuit

The `T05_3DICE_Integration.mo` Modelica file contains several models, that represent various cooling circuits and heatsink models. These components are taken from different libraries within both 3D-ICE and Modelica software, therefore, in order to run a simulation they must be loaded in a correct way. An user must load first the Computer Cooling library package from the repository folder and, after that, the ThermalBlocks and the HeatSinkBlocks. These two libraries contain the 3D-ICE ports required to model the heatsink and are located within `./3d-ice/heatsink_plugin/common/libraries` and `./3d-ice/heatsink_plugin/common` respectively.

After that users should load `T05_3DICE_Integration.mo` in the Modelica environment and choose the system they want to simulate modifying the `T05_Interface3DICE.mo` model. In addition, users can modify or implement other cooling circuits using all the components provided by the Modelica library.

To simulate correctly a cooling circuit and its heatsink, the loaded libraries must be the following:

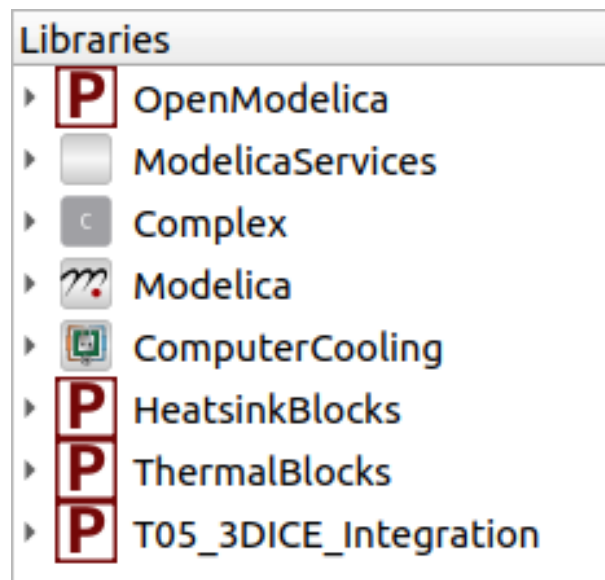


Figure 3.3: Modelica libraries required

3.2.3. Results

At the end of a simulation, some new files will have been created within the `results` folder according to the instruction written inside the `stack` file. The outputs of our default simulation are two different files: The first one is a `trace.txt` file that contains temperature data measured in a single point of the processor, specified by the `stack` file, during the simulation. The second output is a `map.txt` file that prints the temperature map at the end of the simulation in a precise layer of the IC component. Two more files indicating x and y axis are created to be used during the map plotting. In addition, another result file can be added, it is by default called `data.csv`, it holds all the simulation data requested by the user in the heatsink Modelica file, through the Recorder utility.

3.2.4. Scripts

In order to better analyze the simulation results the Computer Cooling library provides some scripts, that can be run using `Scilab`. These are located within the `Scripts` folder and their function is the following: The `plot_transient.sce` plots the temperature transient, it takes the values of time and temperature from `trace.txt`. It can be used in order to be sure that the transient has gone and the temperature has reached the steady state value. The `plot_temperature_map.sce` script allows users to see a map of the final temperature reached by the processor, the temperature values are represented with different colours that makes the analysis more clear. Within this script a function is defined, called `virtualsensors`, which allows users to stamp the temperature value anywhere inside the processor component.

3.3. Example

In the last section of this chapter an example will be introduced in order to show the capability of the 3D-ICE software to simulate a heatsink model developed using the Computer Cooling library.

3.3.1. Modelica scheme

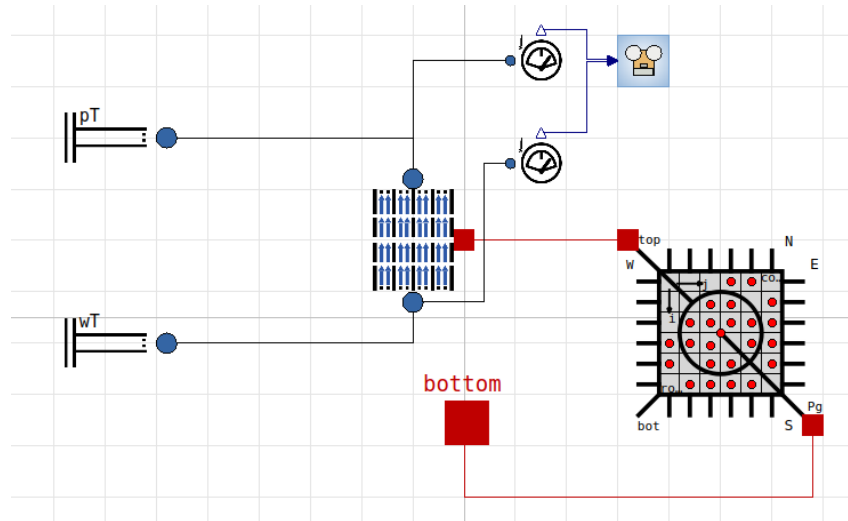


Figure 3.4: Heatsink Modelica model

The Modelica model can be divided into two main parts; the first one is composed by the waterblock component that takes heat away from the processor, it is connected to the rest of the cooling circuit via two boundary condition blocks. All these components are taken from the Computer Cooling library. The second part is composed by the heatsink itself, this model comes from 3D-ICE libraries. In addition, two temperature sensors and a recorder are added to the scheme in order to measure and record the temperature at the beginning and at the end of the waterblock component.

Cooling circuit

In this example we will focus on the waterblock component and the heatsink, rather than the whole cooling system. For this reason the cooling circuit is not represented with all the components but it is defined only by boundary condition values. An user can configure all the parameters of the circuit, in this example the waterblock is fed by water at 24°C with a mass flow rate of 0.12 l/min. The number of liquid channels and number of finite volumes per channel of the waterblock is imposed by 3D-ICE (using the default value of 10) and each channel has a dimension of 3x3cm.

Heatsink

The heatsink component is taken from the 3D-ICE HeatsinkBlocks, it is made of copper and the dimension is the same of the waterblock, 3x3cm.

Sensors and recorder

In order to measure the temperature at the beginning and at the end of the waterblock two sensors are placed and a recorder component is used to store the values in a file called `data.csv` with a sampling time of `0.01s`.

3.3.2. 3D-ICE parameters

The heatsink Modelica model is connected, by means of the 3D-ICE co-simulation interface, with the 3D-ICE model of a square chip with a side of `1.024cm`. The `Stack` and `Floorplan` files are defined as follows.

Stack

In the `Stack` file the dimensions of the chip are defined, It is also possible to set the resolution of the simulation through the value given to the cell discretization, for this example the cell dimension is set to `0.032cm` in order to obtain a high resolution simulation. The die is made of silicon and it has a thickness of `0.61cm`, the simulated time is `25s` with a transient step of `0.015s`. The initial temperature of the chip, heatsink and waterblock component is set to `27°C`.

Floorplan

The `Floorplan` file defines a grid of `4x4` heat generation areas, Each area is splitted into two parts so as to put a temperature sensor in the middle of each heater, the two parts are not individually controllable and generates in total a power of `7.5W`. In the same way, all the heaters generate `7.5W`, with the exception of the second heat generation column that is turned off (i.e. Generates `0W`). With this structure the chip that represents the processor generates a total power of `90W`.

3.3.3. Results

Temperature transient

In order to check that the temperature inside the processor has reached the steady state value it is necessary to collect and plot the temperature trace at the center of the die gathered into the file `trace.txt`. The following figure shows that, at the end of the simulation (i.e. `25s`) the transient is over and so the system has reached the steady state value.

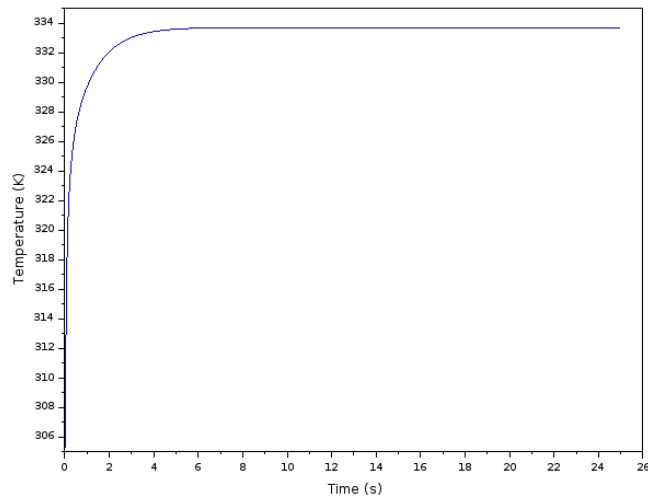


Figure 3.5: Temperature transient at the center of the die

Temperature map

With the temperature map represented in figure 3.6 it is possible to notice that the heating pattern is coherent to the structure defined before. In this example the maximum temperature of 85.2°C is observed in the top right heating elements, in fact it can be observed that the temperatures in the bottom part of the chip are lower, this is due to the fact that the water inlet is located there. If we analyze the second column in which the heaters are turned off it is possible to see that, consistently what said before, the temperature at the bottom is lower.

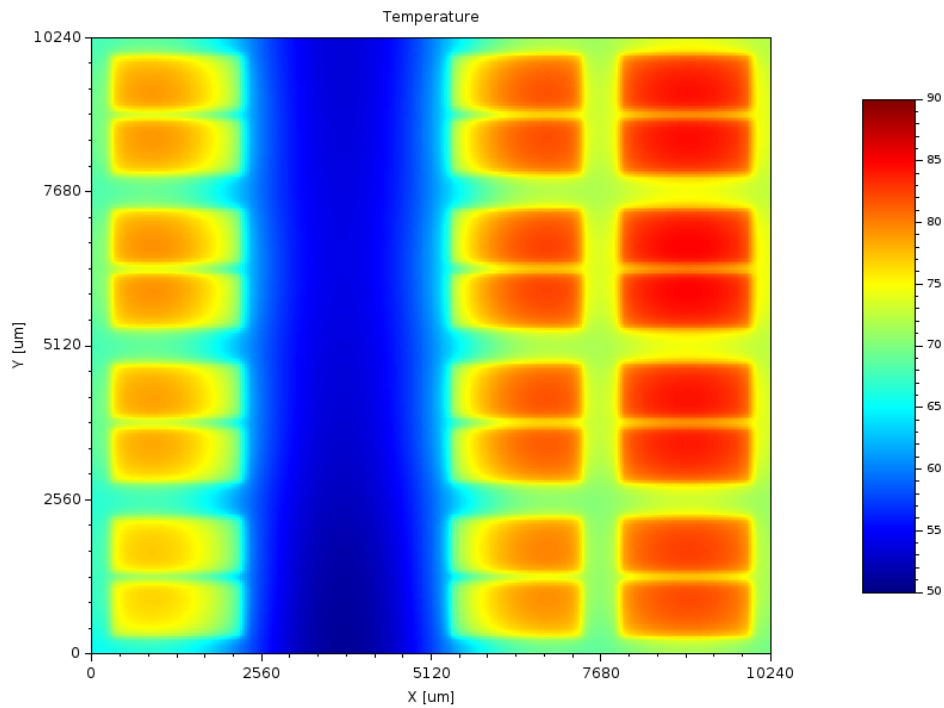


Figure 3.6: Temperature map at the end of the simulation

Temperature recorded

From `data.csv` file it is possible to plot the temperature at the beginning and at the end of the waterblock. In this example makes no sense to represent those values since the temperature at the inlet is fixed to 24°C by boundary conditions and the whole cooling circuit is not modelled. The recorder is added to the example only to show its functioning.

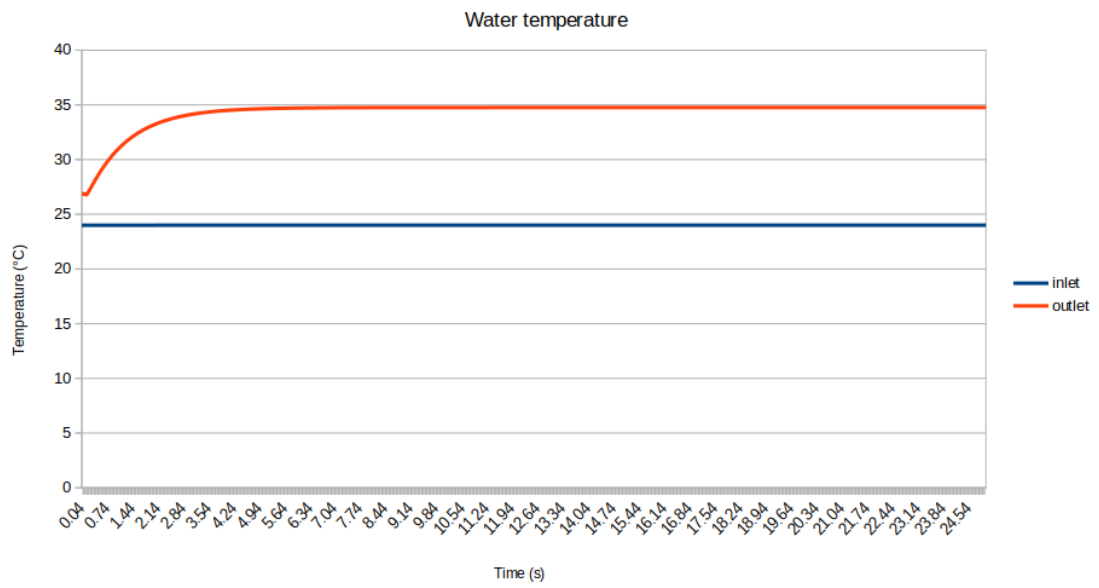


Figure 3.7: Water temperature at the inlet and outlet of the waterblock

4 | Example of a rack system

In this last chapter an example of a liquid cooled rack system architecture will be presented. All the models used to set-up and simulate this liquid cooling system are taken from the Modelica library developed within this work. With this experiment is therefore possible to demonstrate the capabilities of the library and the importance of simulations in the search for an optimal implementation design.

First, an overview on the rack system architectures will be explored so as to explain the main components and structures that will be implemented later on [21, 29]. The experimental set-up will be shown afterward, both from the liquid cooling system side, the rack system, and the controller side. After that, simulation results will be presented and some parameters will be analyzed and modified so as to show the effect of some variations in the whole system. Lastly, the number of CPU units will be increased in order to evaluate the capability, and performances, of the library to simulate large scale models.

4.1. Liquid cooled rack system architecture

A liquid cooled rack system can be divided into three main parts: a CDU, a hot and cold manifold and several CPU units also called server loops [21]. The figure 4.1 shows the whole architecture divided into the three parts, the detailed description of its component is presented later.

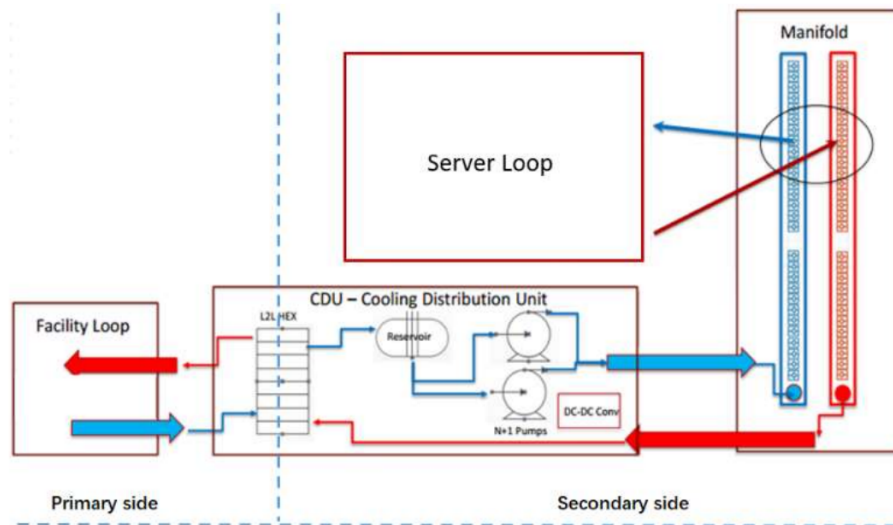


Figure 4.1: Liquid cooled rack system topology [21]

The coolant distribution unit (CDU) is a specific equipment that allows heat transfer between the server and the external environment. The CDU consists of a set of interfaces, pumps, heat exchangers, reservoir tanks, valves, controllers and sensors that guarantee the correct flowing of the liquid inside the cooling circuit. The CDU controls the water supply into the CPU units, this control action is based on the external ambient temperature and on the IT workload within server loops. Usually the CDU provides a real-time communication through Human-Machine Interaction, the operating parameters, the status of the main equipment and any alarm signal can be transmitted to an host computer so as to coordinate simultaneously different CDU units.

Another key component inside a rack architecture is the manifold system, its purpose is to distribute the cooling liquid within the rack to and from the IT equipment (CPU unit).

Analyzing the path of the liquid inside a cooling architecture, starting from the reservoir tank within the CDU, the cold liquid is pumped inside the cold manifold and enters the CPU units, where waterblock components allow the heat exchange with the IT equipment. Then, the outlet hot liquid goes through the hot manifold and reaches the CDU heat exchanger, a radiator, that cool down the liquid to the ambient temperature.

More details on the components are presented in the following where an example of a rack system is designed using the Computer Cooling Modelica library.

4.2. System set-up

In order to present clearly the components used to set-up this architecture in the Modelica environment, the system is subdivided into three main parts: The liquid cooling system (CDU), the rack system (manifolds and server loop) and the control system that usually is implemented in the CDU unit, as said before. In this example the system components are dimensioned in order to refrigerate three CPU units, the model diagram in the figure 4.2 shows the whole architecture within the Modelica environment.

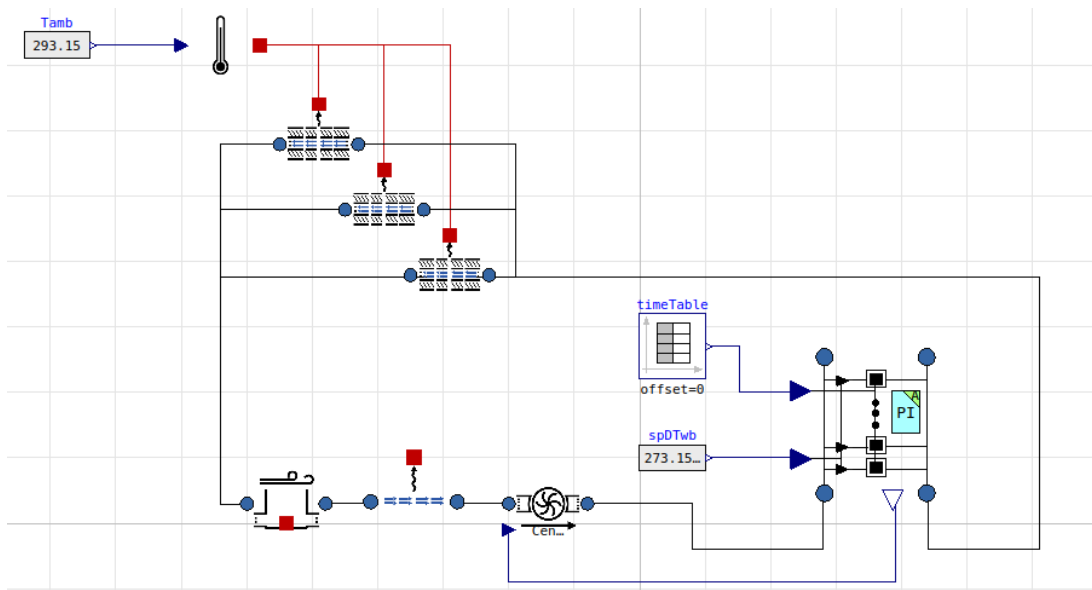


Figure 4.2: System architecture within Modelica

4.2.1. Liquid cooling system

The liquid cooling system is composed by several components that make possible the flowing of the cooling liquid, in our case water, into the rack system. As said before, all these components are located into the CDU unit, in our Modelica architecture these components are placed out of the `CPU_Array` model, because they do not need to be replicated in case of an increasing of CPU units. There is only a unique CDU unit inside our architecture and below are listed and analyzed the components present inside.

Radiator

The radiator is a key component inside the liquid cooling circuit, its purpose is to allow the heat exchange between the water and the external environment. The heat exchanger of a CDU unit can be an air-to-liquid system, where a set of fans is used to cool down

the water of the circuit, or it can be a liquid-to-liquid system in which an external liquid circuit refrigerates the medium.

To represent the forced air convection of the radiator, three `Tube_1D` models have been implemented, each of them has a length of 1m, a stream diameter of 1cm and a thickness of 1mm. The nominal pressure difference is 500Pa and the nominal mass flow rate is 5 l/min for each `Tube_1D` model. The ambient temperature for this experiment is set to 20°C.

Reservoir tank

Every liquid cooling system is designed with a pressure stabilization system to avoid pressure fluctuation or even pipe damage due to thermal expansion and contraction of the coolant, the stabilization system is also useful to avoid negative pressure at the inlet of the components due to pressure drop through the pipeline. In our example we have implemented a vented tank that works as a pressure stabilization system and has a capacity of 1 l, we do not take into consideration the heat dissipation of the water inside the tank, for this reason the heat port is not connected to the external environment boundary condition.

Duct

To represent the pressure drops along the liquid circuit inside the CDU a `LiquidStream` model has been implemented, this model, as in the previous case, do not take into consideration the heat dissipation of the fluid because it is negligible with respect to the other system components, such as the radiator and the waterblock. For the experiment we have considered a duct diameter of 1cm and a length of 3m. The nominal pressure difference is 500Pa and the nominal mass flow rate is 15 l/min, based on the system dimension.

Pump

The Pump is the heart of the system, it provides liquid flow to other components. The pump used for our experiment is a high-speed centrifugal pump with a nominal pressure difference of 0.15bar in case of zero flow and 0.1bar in nominal flow working condition. The pump is dimensioned in order to supply water to three CPU units, for this reason it has a nominal flow rate of 15 l/min. In order to coordinate the action of the pump and the opening of the valves, this component is controlled by all the PIs in a specific manner that will be presented later on.

4.2.2. Rack system

To correctly model the server loops and the manifold distribution system a component named `CPU_array_with_PI_controllers` has been developed. This model allows users to set the number of CPU units composing the rack system, all these units are arranged in parallel and controlled independently. Since the model can represent a single or multiple CPU units, all the components instantiated inside can be duplicated.

An user can set the parameters of each component within the rack such as the liquid streams, the valves and the CPU units. It is important to point out that all the values are the same for all the componets that will be duplicated, i.e. if we set the spreader thickenss dimension, this value is the same for all the spreaders of the CPU units. All the component involved are listed and analyzed below.

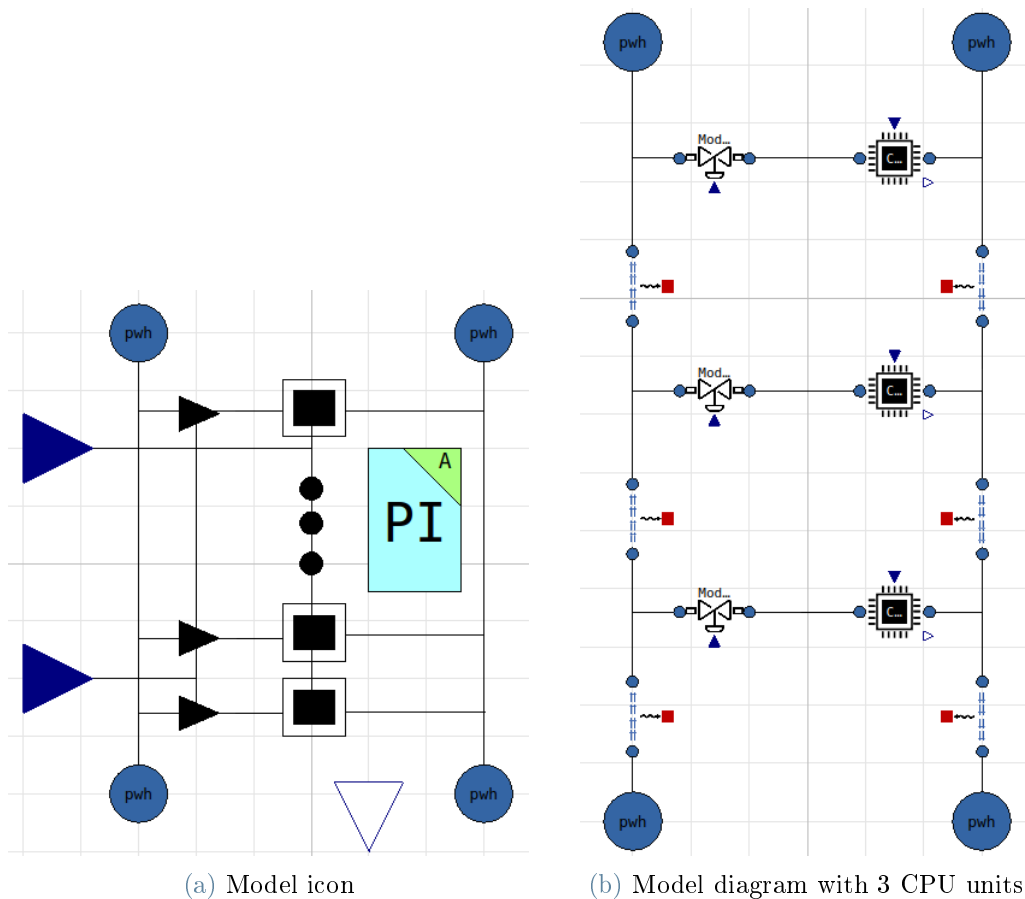


Figure 4.3: `CPU_array_with_PI_controllers` model representing the rack system

Manifolds

The `CPU_array_with_PI_controllers` has four hydraulic ports that represent the inlet and the outlet of the cold and hot manifolds. In order to model the pressure drops inside the manifold system two `LiquidStream` components has been added for each CPU unit, one in the cold part and one in the hot part. An user can set the length and the diameter of a single section of the manifold together with the nominal pressure difference and mass flow rate, it is also possible to set how to subdivide the component along its length, setting the number of volume lumps. As before, the heat dissipation inside the `LiquidStream` is not taken into account because it is negligible with respect to the CPU waterblocks and other components.

For this example we have considered a `LiquidStream` length of 1m and a diameter of 5mm. Considering the nominal mass flow rate, even if the flow is subdivided into each CPU unit, we have to scale the manifold according to the dimension of the whole system, in order to provide the same amount of water to all the CPU units. The nominal mass flow rate is, therefore, 15 l/min, the nominal pressure difference is set to 500Pa coherently with all the other stream components.

Valve

To control independently the flow rate inside each CPU unit a valve has been implemented. Each valve is designed to provide the same amount of flow rate inside each waterblock within the CPU unit, for this reason, in our example the modulating valve has a nominal mass flow rate of 5 l/min, coherently to the fact that we have implemented three CPU units, the nominal pressure difference is set to 5000Pa.

Each valve is controlled via a PI and its control action is based on the temperature on the CPU surface, more details about the control strategy and its implementation will be discussed later on.

CPU unit

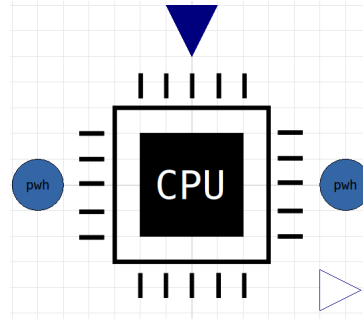


Figure 4.4: CPU_unit icon view

The `CPU_unit` model contains several components used to represent the processor, its spreader and a waterblock that simulate the heat exchange between the cooling liquid and the IT equipment.

First, a prescribed power component has been implemented to represent the internal power generated by the CPU, its value can be set by the user externally and it refers to each CPU unit.

The CPU and the spreader are represented via two `PlanarWall_MultiLayer_UniformGrid` components, an user can set the geometrical dimensions and the number of lumps along the three directions. For this example we have considered a squared CPU of 2cm with a thickness of 1mm and a spreader of 5cm with a thickness of 3mm.

The last key element of the `CPU_unit` model is the waterblock, this component allows the heat exchange between the IT equipment and the cooling water, its heat port is connected to the external side of the spreader. The waterblock component has also two hydraulic ports that represents the inlet and the outlet of each CPU unit. For this example we have considered a waterblock with 4 liquid cannels, each with a length of 5cm. The nominal mass flow rate and pressure difference are set coherently to 5 l/min and 500Pa respectively.

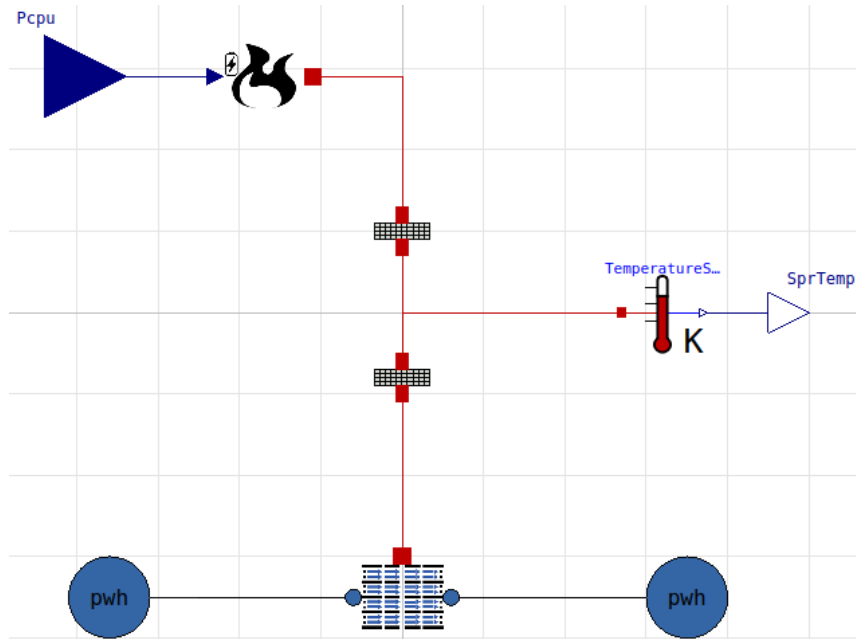


Figure 4.5: CPU_unit diagram view

4.2.3. Control system

To correctly control the whole system a PI controller has been developed for each CPU unit, for this reason the PIs are implemented inside the `CPU_array_with_PI_controllers` model in order to be replicated consistently with the number of CPU units.

Each controller compute the control action taking as input the temperature of the internal side of the spreader, the one that is directly connected to the CPU (the sensor location can be seen in the figure 4.5), this control action is used to command the respective modulating valve opening. The control structure within the model is represented in the figure 4.6.

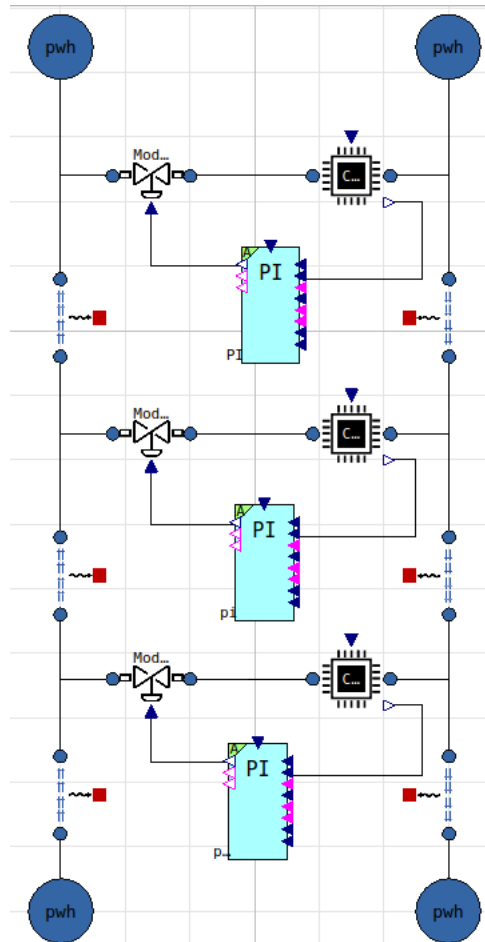


Figure 4.6: Model diagram with 3 CPU units and their controllers

In order to control also the external pump in a coordinated fashion, all the valves control actions are added up in a weighted manner. The basic idea behind this strategy is to avoid too much stress on the pump and valve components, that may occur when the two actuators works against each other. The total control action is computed inside the `CPU_array_with_PI_controllers` model and it is presented as an output connector so as to be linked to the pump command signal.

For this example a simple PI controller has been implemented, where its parameters are tuned with trial-and-error method, to reach a fast response without overshoot and oscillations. More complex control structures and strategies can obviously be implemented but this is not the focus of this work. The final controller has a gain of 0.1 and a time constant of 2s, the minimum value for the valve opening is set to 0.02 in order to avoid the growing of bacteria or algae in the liquid, that may occur when the flow is not permitted and the coolant water stops inside the components.

4.3. Simulation results

Before showing the simulation results some more details about the boundary conditions and the simulation setup will be presented. The ambient temperature, the one that cool down the water inside the radiator is set to 20°C. To simulate the heating action of the IT equipment a power trace table has been implemented, the power generated by the processor oscillates between 5W and 80W. The maximum temperature limit allowed by the PI controllers is 40°C for all the CPU units. Considering the simulation setup we simulate a 3-hour-long working session with a sampling time of 3s.

The first simulation result that is important to analyze is the CPU temperature during the complete working session, it is represented in the figure 4.7 together with the power generated by the CPU itself.

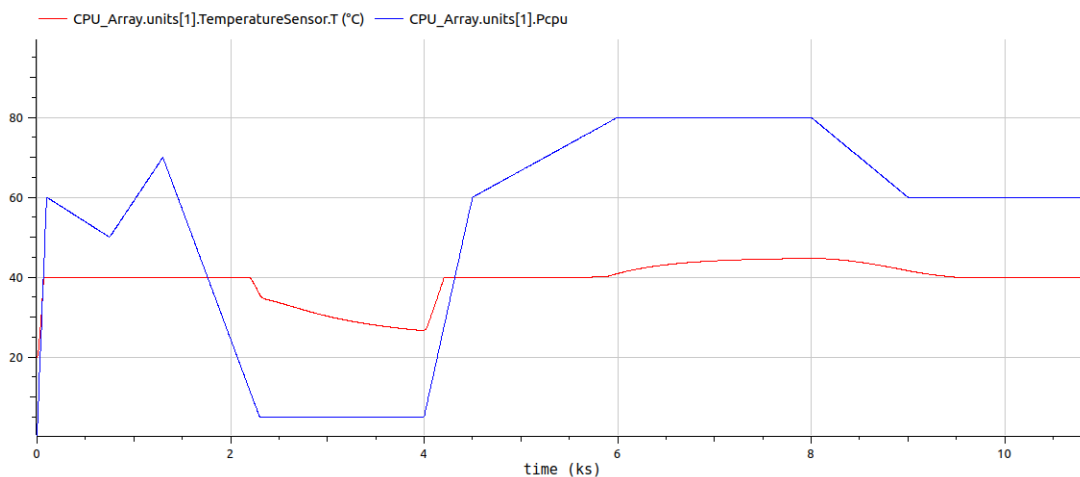


Figure 4.7: Simulation results: CPU temperature and power generated

It is worth noticing that the figure represents the temperature and the power generated by the first CPU unit, but since the system is correctly dimensioned, all the CPU units has the same temperature dynamic, in fact, the inlet water conditions of each CPU unit is the same.

Analyzing the graphical simulation results it is possible to notice that once the CPU temperature has reached the maximum allowable value set by the controller (i.e. 40°C), the control system is activated and the valve starts to open (figure 4.8), allowing the fluid to pass through the waterblock (Figure 4.9), cooling down the IT equipment. In the same way the pump increases the speed providing a higher mass flow rate at higher pressure to the waterblock. This equivalent behaviour of the two actuators shows that the control strategy allows a coordinate utilization of the valves and the pump.

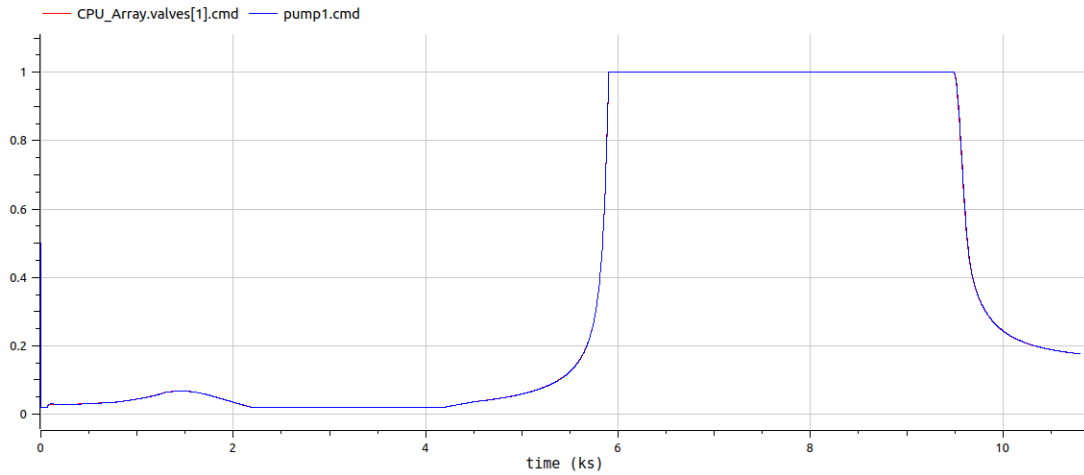


Figure 4.8: Simulation results: pump and valve command

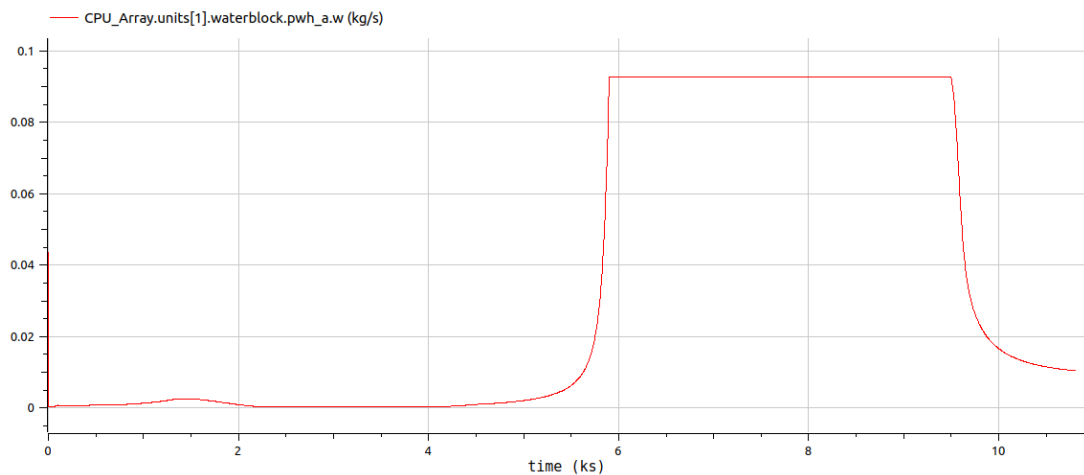
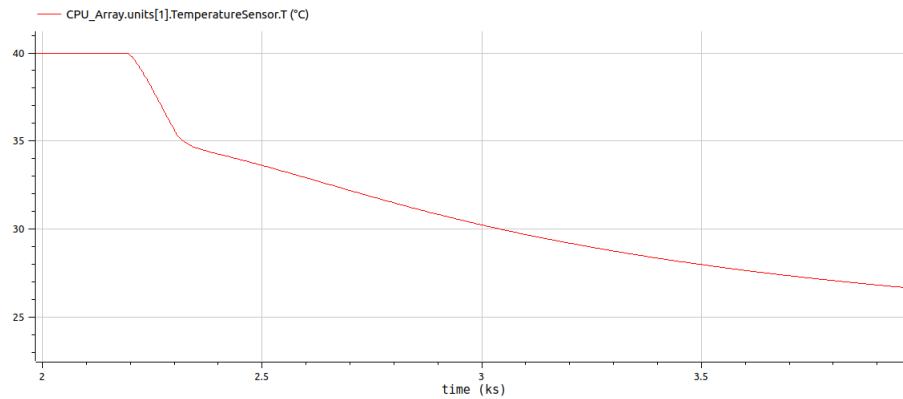
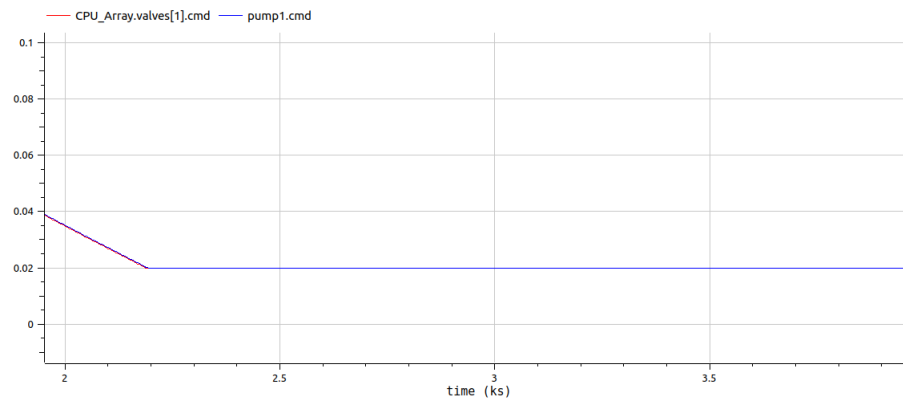


Figure 4.9: Simulation results: waterblock inlet mass flow rate

There are two main dynamics that are worth noticing, The first is the temperature decreasing after 2000s, a detailed graphical representation can be seen in the figure 4.10a. The power generated by the processor decreases to 5W, now the controller can close the valve to the minimum value and slow down the pump (Figure 4.10b), this ends up with a temperature decreasing dynamic divided into two steps; during the first one the temperature decreases faster because the valve and the pump can be closed and slowed down respectively, but when they reach the minimum value it is no more possible to reduce the flow rate and the water pressure, so the IT equipment and the water cool down slowly, according to the thermal characteristic of the components themselves.



(a) Temperature detail



(b) Valve and pump command detail

Figure 4.10: Simulation results: decreasing temperature dynamic in detail

The second important behaviour can be pointed out after 6000s (figure 4.7), the power generated by the CPU reaches its maximum value and the controller is no more able to maintain the temperature below the threshold of 40°C. This is due to the saturation of the two actuators, in fact when the valve reaches its maximum opening and the pump operates at maximum speed there is no way to cool down the IT equipment. Analyzing the temperature of the water at the inlet and at the outlet of the waterblock (figure 4.11) it is possible to notice that, when the system saturates, the two temperatures are closer, this means that the cooling system is no more able to refrigerate the IT equipment. This behaviour can be observed also analyzing the water temperature at the inlet and outlet of the radiator component, as shown in figure 4.12.

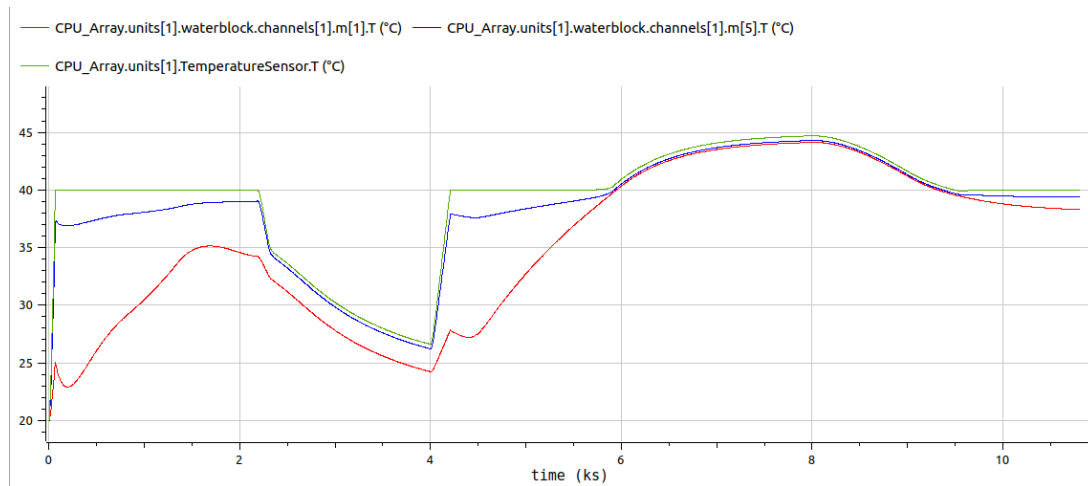


Figure 4.11: Simulation results: waterblock inlet/outlet water temperature

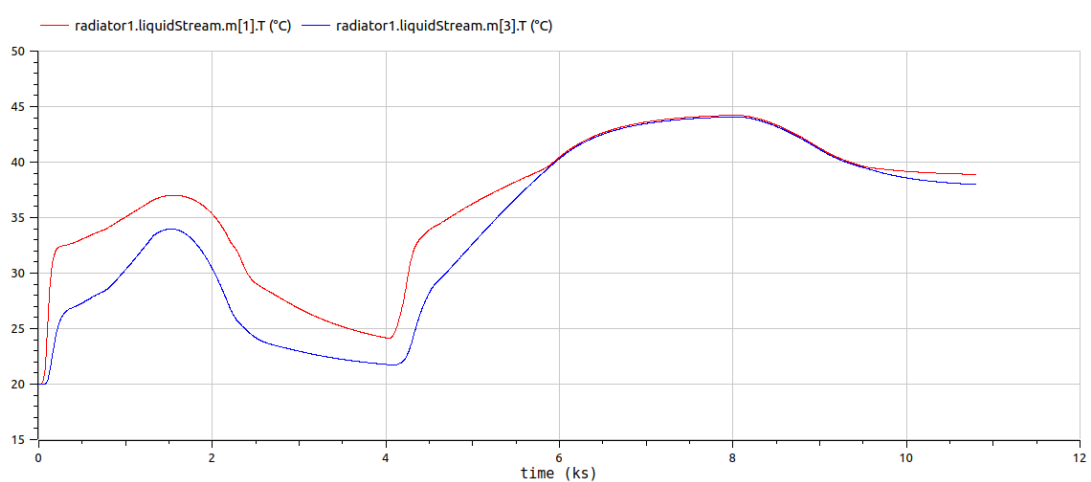


Figure 4.12: Simulation results: radiator inlet/outlet water temperature

4.4. Variations

To show the capability of the Computer Cooling library and to demonstrate the importance of simulations in the search for an optimal implementation design, in this section some parameters and dimensions are slightly modified and a few other simulations are proposed.

Larger radiator dimensions

As a solution to the previously discussed saturation problem we now consider a larger radiator system, more precisely we increase the diameter of the radiator tubes from 1cm to 3cm and we extend the length of the tubes from 1m to 1.2m, with this new configuration

the heat exchange between the cooling water and the environment increases, this means that the radiator component can cool down the liquid in a more effective way. The result of these variations can be seen in the figure 4.13 where the controllers can maintain the temperatures below the threshold and the saturation does not occur anymore.

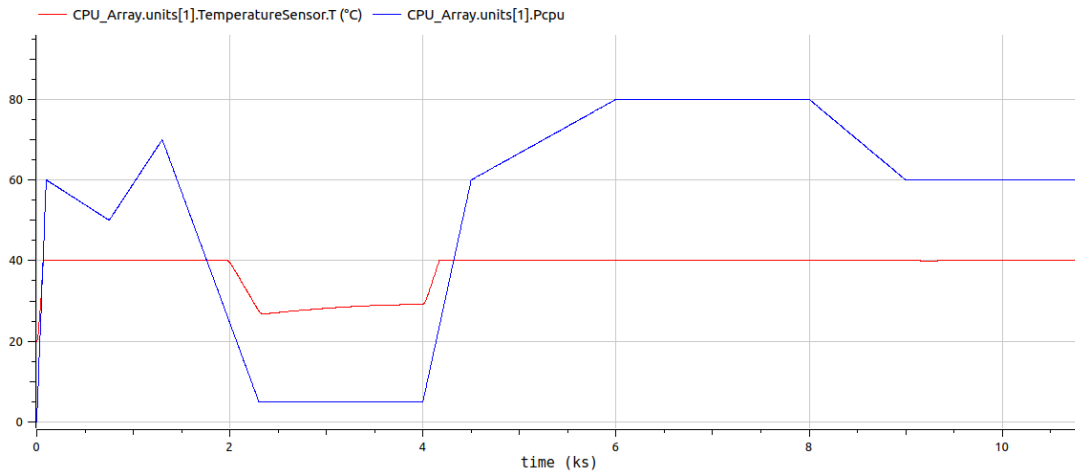


Figure 4.13: Simulation results: CPU temperature and power generated

Higher number of CPU units

In the last part of this chapter the performance of the library will be tested, as said before the `CPU_array_with_PI_controllers` model has been developed as an expandable component, this means that in order to increase the number of CPU units it is sufficient to modify the corresponding parameter. It is obvious that also the cooling system must be adapted to the new dimension of the rack system, this aspect is not the focus of this test and so it is not treated here. The purpose is, instead, to collect some data so as to evaluate the performance and the smart implementation of the library components. To do so a 3-hour-long working session will be simulated and the number of CPU units will be increased continuously.

All the simulations are performed on an AMD Ryzen 7-5800H running Ubuntu 20.04 using OpenModelica 1.20.0 dev-319-gbd71b95.

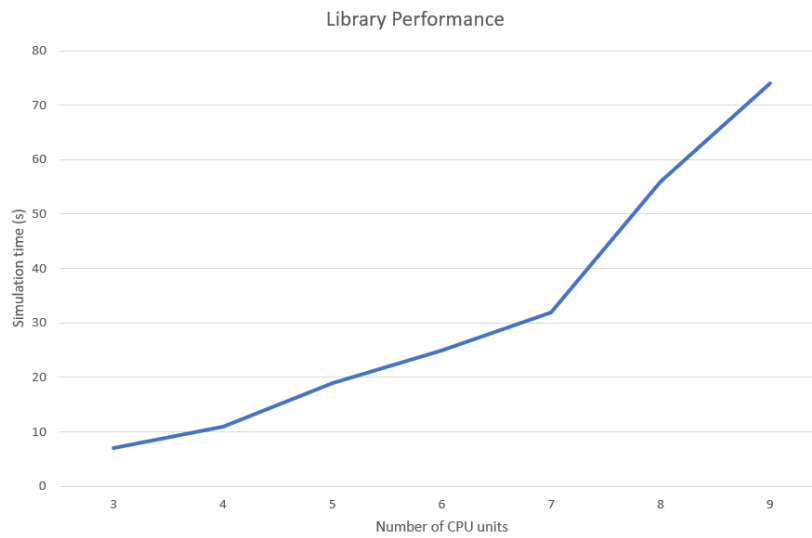


Figure 4.14: Library performances

The graphical result shows that the 3-hour-long simulation took only 7s in the case of 3 CPU units, but also in the case of 9 CPU units, with 3x the number of equations, the time required to simulate the whole working session is 74s which is approximately 146x faster than real time, it is therefore possible to conclude that the performances of the library are quite good and the models has been implemented in an efficient way.

5 | Conclusions and future developments

In this thesis an analysis on several computer cooling systems was carried out. To avoid the overheating problem of the electrical components various strategies and technologies can be applied, in this work the advantages and drawbacks of different approaches has been discussed, also considering the choice of the modelling and simulating platform.

The main purpose of this work is to provide a new set of tools in order to study, design and simulate innovative and emerging cooling strategies, for both the private and the industrial setting.

To do so a pre-existent modelica library has been improved and extended, several new models are added to better represent various cooling systems for a wide range of uses. This work focuses more on the liquid cooling solutions, that are capable of supporting high density power and offer multiple advantages, recalling that to improve energy efficiency is crucial, not only to allow an industry growth but also to reduce operational costs.

Another important tool that has been studied is 3D-ICE, a simulation platform written in C, already existent and able to quickly simulate an IT equipment in a very detailed way. This thesis provide a template that can be used in order to easily connect 3D-ICE and the Modelica library.

Two different examples has been developed, one for each category of usage (private and industrial), to show the completeness and the capability of the library. The Heatsink model is also used to demonstrate the user friendly interface between Modelica and 3D-ICE, whereas the rack architecture was an optimal opportunity to evaluate also the performance of the simulations.

Concerning the future development it is clear that the Modelica library can be expanded further; new components, media and solid materials can be easily added to the library following the same structure of the existent models. With the implementation of new components belonging to different physical domains, for example ideal gas and moist air

components, more complex systems can be modelled and simulated. This multi-domain capability of the Modelica library can be the next step in the development of the library itself. Another important improvement can be done about the interaction between 3D-ICE and the library, the combined simulation can be accelerated and optimized, one of the major bottleneck is caused by the low accessibility of Modelica variables when simulating from 3D-ICE. To solve this problem a new software tool can be developed.

We hope that the Modelica library presented within this work can become a container of simulation models for innovative and emerging cooling systems applied to computer environment and, more in general, to electronic components environment.

Bibliography

- [1] X. Zhang, T. Lindberg, N. Xiong, V. Vyatkin, and A. Mousavi, “Cooling energy consumption investigation of data center it room with vertical placed server,” *Energy procedia*, vol. 105, pp. 2047–2052, 2017.
- [2] Y. Zhang, Y. Zhao, S. Dai, B. Nie, H. Ma, J. Li, Q. Miao, Y. Jin, L. Tan, and Y. Ding, “Cooling technologies for data centres and telecommunication base stations—a comprehensive review,” *Journal of Cleaner Production*, vol. 334, p. 130280, 2022.
- [3] Q. Zhang, Z. Meng, X. Hong, Y. Zhan, J. Liu, J. Dong, T. Bai, J. Niu, and M. J. Deen, “A survey on data center cooling systems: Technology, power consumption modeling and control strategy optimization,” *Journal of Systems Architecture*, vol. 119, p. 102253, 2021.
- [4] A. Capozzoli and G. Primiceri, “Cooling systems in data centers: state of art and emerging technologies,” *Energy Procedia*, vol. 83, pp. 484–493, 2015.
- [5] H. Rong, H. Zhang, S. Xiao, C. Li, and C. Hu, “Optimizing energy consumption for data centers,” *Renewable and Sustainable Energy Reviews*, vol. 58, pp. 674–691, 2016.
- [6] M. J. Ellsworth Jr and M. K. Iyengar, “Energy efficiency analyses and comparison of air and water cooled high performance servers,” in *International Electronic Packaging Technical Conference and Exhibition*, vol. 43604, pp. 907–914, 2009.
- [7] “The benefits of liquid cooling over air cooling for power electronics,” 2011.
- [8] A. H. Khalaj and S. K. Halgamuge, “A review on efficient thermal management of air-and liquid-cooled data centers: From chip to the cooling system,” *Applied energy*, vol. 205, pp. 1165–1188, 2017.
- [9] M. K. Patterson and D. Fenwick, “The state of data center cooling,” *Intel Corporation White Paper*, 2008.
- [10] R. Sinha, C. J. Paredis, V.-C. Liang, and P. K. Khosla, “Modeling and simulation methods for design of engineering systems,” *J. Comput. Inf. Sci. Eng.*, vol. 1, no. 1, pp. 84–91, 2001.

- [11] Y. Fu, W. Zuo, M. Wetter, J. W. VanGilder, X. Han, and D. Plamondon, “Equation-based object-oriented modeling and simulation for data center cooling: A case study,” *Energy and Buildings*, vol. 186, pp. 108–125, 2019.
- [12] K. Hinkelman, J. Wang, W. Zuo, A. Gautier, M. Wetter, C. Fan, and N. Long, “Modelica-based modeling and simulation of district cooling systems: A case study,” *Applied Energy*, vol. 311, p. 118654, 2022.
- [13] A. Agrawal, M. Khichar, and S. Jain, “Transient simulation of wet cooling strategies for a data center in worldwide climate zones,” *Energy and Buildings*, vol. 127, pp. 352–359, 2016.
- [14] S.-W. Ham and J.-W. Jeong, “Impact of aisle containment on energy performance of a data center when using an integrated water-side economizer,” *Applied Thermal Engineering*, vol. 105, pp. 372–384, 2016.
- [15] S. E. Mattsson and H. Elmqvist, “Modelica-an international effort to design the next generation modeling language,” *IFAC Proceedings Volumes*, vol. 30, no. 4, pp. 151–155, 1997.
- [16] J. L. Hensen and R. Lamberts, *Building performance simulation for design and operation*. Routledge, 2012.
- [17] T. Cancelliere, “Object-oriented modeling and simulation of datacentre and hpc cooling,” Master’s thesis, Politecnico di Milano, 2020-2021.
- [18] A. Sridhar, A. Vincenzi, D. Atienza, and T. Brunschwiler, “3d-ice: A compact thermal model for early-stage design of liquid-cooled ics,” *IEEE Transactions on Computers*, vol. 63, no. 10, pp. 2576–2589, 2013.
- [19] F. Terraneo, A. Leva, W. Fornaciari, M. Zapater, and D. Atienza, “3d-ice 3.0: efficient nonlinear mpso thermal simulation with pluggable heat sink models,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1062–1075, 2021.
- [20] J. Gullbrand, M. J. Luckeroth, M. E. Sprenger, and C. Winkel, “Liquid cooling of compute system,” *Journal of Electronic Packaging*, vol. 141, no. 1, 2019.
- [21] S. W. G. C. Q. F. J. Z. N. A. Y. X. H. Z. J. Y. Q. Q. Ruiyu Sun, Yongwei Li, “White paper: an advanced liquid cooling rack design for data center,”
- [22] “Modelica association,” 2021.
- [23] H. Elmqvist, S. E. Mattsson, and M. Otter, “Modelica: The new object-oriented

- modeling language,” in *12th European Simulation Multiconference, Manchester, UK*, vol. 5, 1998.
- [24] “Modelica libraries,” 2021.
- [25] P. Fritzson, *Introduction to modeling and simulation of technical and physical systems with Modelica*. John Wiley & Sons, 2011.
- [26] M. Tiller, *Introduction to physical modeling with Modelica*. Springer Science & Business Media, 2001.
- [27] G. B. M. R. T. B. M. J. F. T. Arvind Sridhar, Alessandro Vincenzi, *3D-ICE user guide*, 3.1.0 ed., 7 2021.
- [28] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, “3d-ice: Fast compact transient thermal modeling for 3d ics with inter-tier liquid cooling,” in *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 463–470, IEEE, 2010.
- [29] M. Iyengar, M. David, P. Parida, V. Kamath, B. Kochuparambil, D. Graybill, M. Schultz, M. Gaynes, R. Simons, R. Schmidt, *et al.*, “Server liquid cooling with chiller-less data center design to enable significant energy savings,” in *2012 28th annual IEEE semiconductor thermal measurement and management symposium (SEMI-THERM)*, pp. 212–223, IEEE, 2012.

List of Figures

1	Energy consumption distribution of a data centre [5]	1
2	Example of a cooling system	2
3	Shorter caption	3
1.1	Floorplant and stack files [27]	11
1.2	3D-ICE outputs [27]	12
1.3	Diagram of the Starting Computer Cooling library [17]	13
2.1	Stream confinement component (icon view)	18
2.2	Heat propagation through a CylindricalWall_FiniteVolume model	19
2.3	PlanarWall_MultiLayer_UniformGrid component (icon view)	20
2.4	Uniform grid solid component	21
2.5	Volume decomposition via uniform grid	21
2.6	PlanarWall_MultiLayer_StructuredGrid component (icon view)	22
2.7	Structured grid solid component	23
2.8	Tube_1D_Cylindrical model (diagram view)	24
2.9	Heat propagation from external heat port to fluid	24
2.10	WaterBlock_GenericSection_uniform_w component (icon view)	25
3.1	Directories scheme	28
3.2	Contents of MyHeatSink directory	28
3.3	Modelica libraries required	29
3.4	Heatsink Modelica model	31
3.5	Temperature transient at the center of the die	33
3.6	Temperature map at the end of the simulation	34
3.7	Water temperature at the inlet and outlet of the waterblock	35
4.1	Liquid cooled rack system topology [21]	38
4.2	System architecture within Modelica	39
4.3	Shorter caption	41
4.4	CPU_unit icon view	43

4.5	CPU_unit diagram view	44
4.6	Model diagram with 3 CPU units and their controllers	45
4.7	Simulation results: CPU temperature and power generated	46
4.8	Simulation results: pump and valve command	47
4.9	Simulation results: waterblock inlet mass flow rate	47
4.10	Shorter caption	48
4.11	Simulation results: waterblock inlet/outlet water temperature	49
4.12	Simulation results: radiator inlet/outlet water temperature	49
4.13	Simulation results: CPU temperature and power generated	50
4.14	Library performances	51