POLITECNICO DI MILANO

Scuola d'Ingegneria Industriale e dell'Informazione

Master of Science in Automation and Control Engineering

A Test Suite of Large Scalable Models for Modelica Tool Evaluation

Supervisor: Prof. Francesco Casella

Author: Kaan Sezginer
Matricola: 796590

2014-2015

Acknowledgements

First of all, I would like to thank Prof. Francesco Casella for being my supervisor and letting me work on this interesting thesis subject. And, I would also like to thank him for discussing and helping constantly in any kind of difficulty, being patient and supporting me throughout this thesis work.

I feel very lucky having had the experience of being a Master of Science student at Politecnico di Milano. During my studies, I believe that I gained exceptional experiences, unforgettable memories that I will always remember and I leveraged myself to a higher level where I can see the life from a different aspect.

I would like to thank all of my friends whom I met from all around the world. Without them, it would not have been possible to have great moments.

Especially, I would like to thank my family for their endless supports and making everything possible.

Kaan Sezginer

Table of Contents

List of Figures

List of Tables

Sommario

L'obiettivo di questa tesina è stato creare e testare in linguaggio Modelica una libreria di grandi modelli scalabili contenente diversi domini: meccanici, termici ed elettrici. Il dominio meccanico include la trave flessibile e la corda, quello termico include conduzioni e scambiatori di calore, mentre il dominio elettrico include modelli su linee di trasmissione. I modelli sono stati implementati in ambiente OpenModelica usando il linguaggio Modelica. Nonostante si sarebbero potuti utilizzare differenti applicazioni di Modelica è stato adoperato OpenModelica Compiler in quanto ritenuto più conveniente. Il progetto vuole mettere in luce la performance di Modelica Compliler rispettando la scalabilità, la quale significa un numero di equazioni in aumento, e intende contribuire a questi concetti chiave. I modelli sono stati implementati discretizzando le loro equazioni differenziali parziali originali e/ o usando gli strumenti di Modelica Standard Library. Tutti i modelli sono stati discretizzati in N che rappresenta il numero di nodi, segmenti o elementi a seconda dei modelli. I modelli, in funzione di N, permettono che la loro discretizzazione possa essere ingrandita ed è stato possibile paragonare i risultati in dettaglio. Per verificare questi modelli sono state implementate soluzioni analitiche o metodi numerici. In questa tesina sono discussi i grafici dei modelli e sono fornite le prestazioni di OpenModelica Compiler in termini di tempi di compilazione e simulazione per valori di N in aumento. I grafici dei modelli hanno mostrato che, all'aumentare della discretizzazione, essi rispecchiano i risultati attesi. D'altra parte, i tempi di simulazione e compilazione crescono significativamente con l'aumentare della discretizzazione, specialmente nel dominio meccanico. Inoltre è stato osservato che il compilatore e la simulazione sequenziale dovrebbero essere migliorati per supportare grandi modelli. Per rimanere in un limite di tempo ragionevole risultano importanti le strategie come i risolutori sparsi e multi rate o la parallelizzazione.

Abstract

The purpose of this thesis work was to create and test a library of large scalable models in Modelica language which contains different domains: mechanical, thermal and electrical. Mechanical domain includes flexible beam and string, thermal domain includes heat conduction and heat exchanger, and electrical domain includes transmission line models. The models were implemented in the OpenModelica environment using Modelica language. OpenModelica Compiler (OMC) was used because of the convenience, different Modelica applications could also have been used. The work intends to highlight the performance of Modelica compiler with respect to scalability, which means increasing number of equations, in terms of compilation and simulation times, and it intends to contribute these key concepts. Models were implemented by discretizing their original partial differential equations (PDE) and/ or by the tools of Modelica Standard Library (MSL). All the models were discretized into N which is the number of nodes, segments or elements depending on the models. Models were function of N, therefore, discretization of the models could be enlarged and the results were compared in detail. In order to verify the models, analytical solutions or numerical formulas were implemented. In this thesis work, plots of the models are discussed and the performance of OMC are provided in terms of compilation and simulation times for increasing number of N values. Plots of the models have shown that as discretization increases, models reflect the expected results. However, compilation and simulation times grow significantly as discretization increases especially for the mechanical domain. It was observed that compiler and sequential simulation should be improved to support large models. Strategies such as sparse, multi rate solvers or parallelization become increasingly important to stay in reasonable time limits.

1. Introduction

Modelica is an object oriented, declarative and multi-domain language for physical modeling of the complex systems. Object oriented physical modeling requires declarative models which provides the modularity. Therefore, the models of the systems are created by using their equations and the models can be used in different domains while building complex systems. Using connection equations, physical connection between the models are realized. Modularity of Modelica language enables high-reusability of models in different contexts and high-readability of them since its structure is user friendly.

The models were implemented in OpenModelica environment which is an open source Modelica based modeling and simulation environment for industrial, research and teaching usage. For the modeling purposes, OpenModelica Connection Editor (OMEdit) was used which enables modelers to create models both textually and graphically. OMEdit is an open source user interface that provides to build models, connection editing, and simulation of the models and plots of the results. OMEdit communicates with OpenModelica Compiler (OMC) and requests the model information and creates models based on the Modelica annotations.

After the implementation of the models, OMC performs several steps for the simulation of the object oriented models. Firstly, from Modelica source code, parsing of the codes, type checking, class expansion and generation of connection equations are performed. Later, OMC flattens the object oriented models into a system of differential algebraic equations (DAE) and in order to reduce the size of the equations performs optimizations. Furthermore, it reduces index of the system for numerical solutions and minimal set of equations in state-space form are generated. Lastly, sequential C code is generated with a numerical solver to simulate the models.

Object oriented modeling is getting more used and large scalable systems are being tackled using Modelica. However, the test cases present in the MSL are not specifically designed to test the ability of a Modelica compiler to cope with increasingly large models. Existing test suites are not designed to stress the large scale factor. Moreover, increasing complexity and dimensions of the models require parallelization that is partitioning the simulation code into several independent parts using task graph and data dependency graphs. Sequential simulation codes do not provide an effective solution in terms of the simulation time while dealing with some large systems. In this case, parallelization becomes important to be able to stay in reasonable time limits. Furthermore, parallelization algorithms for Modelica applications have been exploited in these days in order to provide speed-ups for multi-core CPUs as sequential codes does not allow speed-ups. Therefore, parallelization have become a growing research area for Modelica applications.

Attempts have been done at generating scalable models to test Modelica compilers such as "*Towards a Benchmark Suite for Modelica Compilers: Large Models"* [14]. The paper stresses

the performance of some Modelica compilers for increasing number of equations which have different structural properties. However, physical modeling is not stressed.

This thesis work was inspired from these reasons and tries to highlight and give more idea about how Modelica responses to large scalable physical models from different domains (different structure of equations) in the case of sequential simulation. In this work, electrical, mechanical and thermal domains were taken into account. To be more specific, for the electrical part, transmission line; for the mechanical part, flexible beam and string; for the thermal part, heat conduction and heat exchanger models were implemented. Depending on the system structure and model, they were implemented by discretizing their constituent equations and/ or using MSL. All the models were function of N which corresponds to number of nodes, number of elements or number of segments in the models. Furthermore, having gradually increased the discretization, therefore N, results were critically investigated. Investigation consisted of analysis of the plots of the models, analysis of the compilation and simulation times, and how OpenModelica responded to the enlarged models from different domains in terms of accuracy and time.

Furthermore, the transmission line model was implemented both by its basic circuit equations and Modelica.Electrical library. And, transmission line circuit models were created in order to test response of the transmission line to a step input in the matched load impedance case. Transmission line models were verified by the calculated time delay between the ends of the transmission line.

The Flexible beam and string models were implemented using Modelica.Mechanics.MultiBody library. As a flexible beam, a cantilever beam was considered and an analytical solution was implemented for the verification. String was considered to be fixed at one end while the other end was considered to be free. And, a pulse was applied from the fixed end. Purpose was to observe a travelling wave along the string model.

For the heat conduction, two different cases were considered and implemented: a rod with fixed temperatures at its ends, and a rod with a fixed temperature at one end while the other end is insulated. They were implemented both by equations and Modelica.Thermal library. Moreover, analytical solutions were implemented for the both cases of heat conduction for verification.

Lastly, two types of heat exchangers: cocurrent and countercurrent heat exchangers were implemented by equations. And, a steady state analysis was performed for the heat exchangers for verification.

All the models were tested with gradually increasing N values. Plots of the results, compilation and simulation times were analyzed according to N values. Therefore, performance of OpenModelica to increasing number of equations from different domains is pointed out in the work.

The thesis consists of two main chapters being Chapter 2 and Chapter 3. In Chapter 2, implementation of the models by their equations, MSL are explained. And, for verification, analytical solutions or numerical formulas are provided. In Chapter 3, simulations of the models are provided for several N values. Simulations are discussed in terms of critical plots, and statistics of compilation and simulation times. And, Chapter 4 concludes the thesis with the comparison of the models and discussion about OpenModelica.

2. Models

In this chapter, models are provided which are Heat Conduction, Heat Exchanger, Flexible Beam, String and Transmission Line. Models are represented by their constituent equations and/ or by the MSL. Analytical solutions or numerical methods are explained for the model verification.

2.1. Heat Conduction

Figure 1: Uniform rod

We considered a uniform rod of length L, height h and width of d. Uniform rod has the density ρ, specific heat capacity c and thermal conductivity λ which were all assumed to be constant. Moreover, the sides of the rod were assumed to be insulated. Two different models were considered in terms of the ends of the rod: in the first model, each end of the rod was exposed to fixed temperatures and in the second model, one end was exposed to a fixed temperature while the other end was insulated. In order to be consistent from now on, the first and the second model will be mentioned as HeatConductionTT (TT stands for two fixed temperatures) and HeatConductionTI (TI stands for fixed temperature and insulated end) respectively.

Both models were implemented both by writing the discretized equations of one dimensional heat equation and by using Modelica.Thermal library. Furthermore, analytical solutions were implemented for the model verification.

2.1.1. Heat Conduction by Equations

We considered a small portion of the rod which has a width of Δx from a distance x, and by considering the conservation of energy the equations were defined. According to the conservation of energy, difference between the heat in from left boundary and heat out from the right boundary has to be equal to the heat change at the portion at Δx in time Δt.

Using Fourier's law and arranging the equations:

$$\frac{ch\rho T(x,t+\Delta t)-T(x,t)}{\Delta t}=\frac{-hd\lambda\frac{\partial T(x,t)}{\partial x}}{\Delta x}+\frac{hd\lambda\frac{\partial T(x+dx,t)}{\partial x}}{\Delta x}$$

Taking the limits Δt and Δx goes to zero, we obtain the PDE of the heat equation;

$$c\rho\frac{\partial T}{\partial t}=\lambda\frac{\partial^2 \mathrm{T}}{\partial x^2}$$

In order to be able to make the definition of the equations in OMEdit, instead of taking the limits we proceeded with the finite differences approximation. We approximated the derivatives of the function by finite difference approximation at discrete points. Therefore, the discretized equations were described in the following form:

$$c\rho\Delta x\frac{dTi}{dt}=\frac{-\lambda(T_i-T_{i+1})}{\Delta x}+\frac{\lambda(T_{i-1}-T_i)}{\Delta x}$$

where $i=2,..,N-1$ and they correspond to temperature nodes along the rod excluding the temperature variables at the ends. Moreover, depending on the boundary conditions we defined T_1 and T_N which are the temperature variables at the ends of the rod.

In HeatConductionTT, T_1 and T_N have constant temperature values. In our model we selected as: T_1=300K, T_N = 330K.

In HeatConductionTI, T_N has a constant temperature value and T_1 is insulated. In our model we selected T_N=300K and T_1 has a boundary condition defined as:

$$c\rho\Delta x\frac{dT_1}{dt}=\frac{\lambda(T_2-T_1)}{\Delta x}$$

Moreover, equations were defined in OMEdit as explained. And, for the definition of the parameters and the variables Modelica.SIunits library were used.

2.1.2. Heat Conduction by Thermal Library

Heat conduction models were implemented using Modelica.Thermal.HeatTransfer-.Components which is under Modelica.Thermal library. By connecting heat capacitors and thermal conductors and assigning appropriate parameter values for each capacitor and each conductor, which are function of N, models were created. In the models, N corresponds to number of nodes. In Figure 2, an example of a heat conduction model is shown.

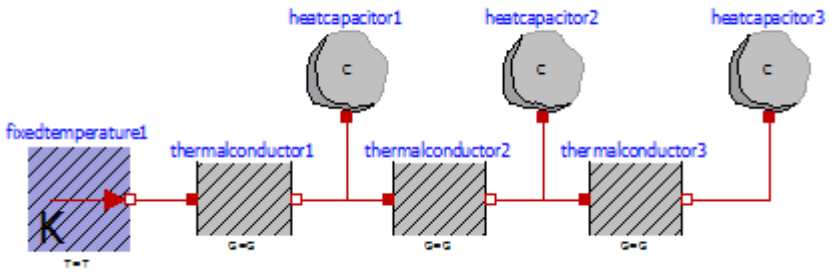

Figure 2: HeatConductionTI example

In Figure 2, a HeatConductionTI model with N=4 nodes is shown, in the same manner a larger model can be created. In the model, at each heat capacitor, a capacitance is defined which varies according to the mass. Moreover, between each heat capacitor a thermal conductor is placed which transports the heat without storing it.

The difference between two models occurred because of the boundary conditions at the ends, thermal conductors and heat capacitors were defined in the same way. In Figure 3, a HeatConductionTT model with N=5 nodes is shown.

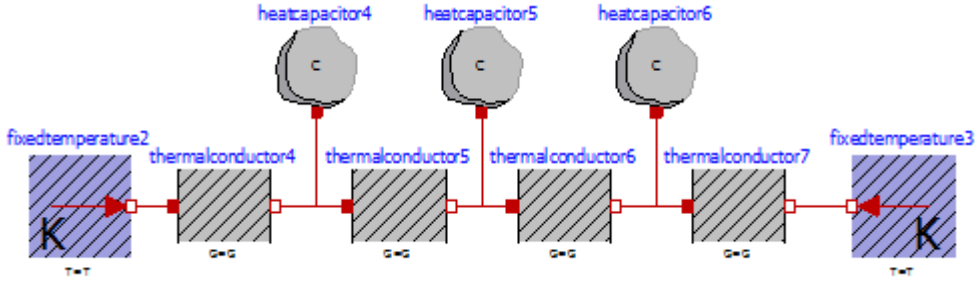

Figure 3: HeatConductionTT example

For the each thermal conductor element, conductance for a box geometry under the assumption that heat flows along the box length was calculated as follows for the both models: $G = \lambda A/l$ where A is the area of box, λ is the thermal conductivity and l is the length of each thermal conductor element. Moreover, heat capacity of the heat capacitors was calculated as follows for the both models: $C = c * m$ where c is the specific heat capacity and m is the mass of each heat capacitor. And, for the parameter and variable definitions Modelica.SIunits library was used.

2.1.3. Analytical Solutions of Heat Conduction Models

Analytical solutions were based on the solutions in "Handbook of Linear Partial Differential Equations for Engineers and Scientists" by Andrei D. Polyanin [4].

If we consider one dimensional partial differential heat equation in the form:

$$\frac{\partial w}{\partial t} = a\frac{\partial^2 \mathrm{w}}{\partial x^2}$$

According to the boundary conditions, solutions for $w(x,t)$ are found.

Solution for the HeatConductionTT:

As mentioned, in HeatConductionTT, the ends are maintained at fixed temperatures. Therefore, the following conditions are prescribed:

$$w = f(x) \quad at \quad t = 0 \ (initial\ condition)$$
$$w = g_1 \quad at \quad x = 0 \ (boundary\ condition)$$
$$w = g_2 \quad at \quad x = l \ (boundary\ condition)$$

where $0 \le x \le l$.

Solution:

$$w(x,t) = \frac{2}{l}\sum_{n=1}^{\infty} \sin(\frac{n\pi x}{l})\exp(-\frac{an^2\,\pi^2\,t}{l^2})M_n(t)$$

where

$$M_n(t) = \int_0^l f(\xi)\sin(\tfrac{n\pi\xi}{l})d\xi + \tfrac{an\pi}{l}\int_0^t \exp(\tfrac{an^2\ \pi^2\tau}{l^2})[g_1 - (-1)^n g_2]d\tau$$

one can transform the solution to:

$$w(x,t) = g_1 + \tfrac{x}{l}\left[g_2 - g_1\right] + \tfrac{2}{l}\textstyle\sum_{n=1}^{\infty}\sin(\lambda_n x)\exp(-a\lambda_n^2 t)R_n(t), \quad \lambda_n = \tfrac{n\pi}{l}$$

where $R_n(t)$ can be written according our boundary conditions:

$$R_n(t) = \int_0^l f(\xi)\sin(\lambda_n\,\xi)d\xi - \frac{1}{\lambda_n}\left[g_1 - (-1)^n g_2\right]$$

Solution for HeatConductionTI:

In HeatConductionTI, one end of the rod is exposed to a fixed temperature while the other end is insulated. At the insulated end, heat flux is zero therefore derivative of the temperature will be zero.

The following conditions are prescribed for the model:

$$w = f(x) \quad at \quad t = 0 \ (initial\ condition)$$
$$\partial_x w = g_1 = 0 \quad at \quad x = 0 \ (boundary\ condition)$$
$$w = g_2 \quad at \quad x = l \ (boundary\ condition)$$

Solution:

$$w(x,t) = \int_0^l f(\xi) G(x,\xi,t) d\xi - a \int_0^t g_1 G(x,0,t-\tau) d\tau - a \int_0^t g_2 H(x,t-\tau) d\tau$$

where

$$G(x,\xi,t) = \frac{2}{l} \sum_{n=0}^{\infty} \cos\left[\frac{\pi(2n+1)x}{2l}\right] \cos\left[\frac{\pi(2n+1)\xi}{2l}\right] exp\left[-\frac{a\pi^2(2n+1)^2 t}{4l^2}\right]$$

$$H(x,t) = \frac{\partial}{\partial \xi} G(x,\xi,t) \Big|_{\xi=l}$$

Therefore, the solutions for each model were defined in OMEdit in order to verify the models and the results are discussed in Chapter 3.

2.2. Heat Exchanger

Heat exchangers are widely used in the chemical process industries. Two fluids flow through the heat exchanger and heat is transferred from hot fluid to the cold fluid. Heat exchangers can be implemented in several ways considering the flow of the fluids through the heat exchanger. In our case, two heat exchangers were implemented which are cocurrent heat exchanger and countercurrent heat exchanger. In cocurrent heat exchanger, fluids flow in the same direction, however, in countercurrent heat exchanger, fluids flow in the opposite directions. Cocurrent and countercurrent heat exchanger models consist of two channels A and B, and a separating heat transfer wall in between. Fluids A and B are flowing in the channels A and B respectively. Heat exchangers were assumed to be insulated around the outside, therefore, heat transfer occurs just between the fluids A and B. Fluid B was considered as the hot fluid and fluid A was considered to be the cold fluid.

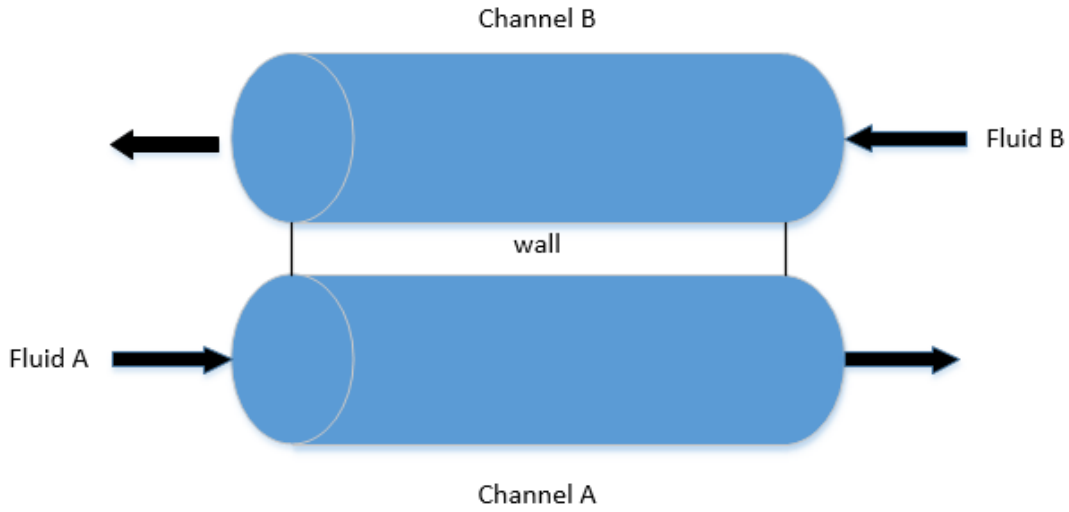

Figure 4: Countercurrent heat exchanger with fluid A and B flowing through the channels

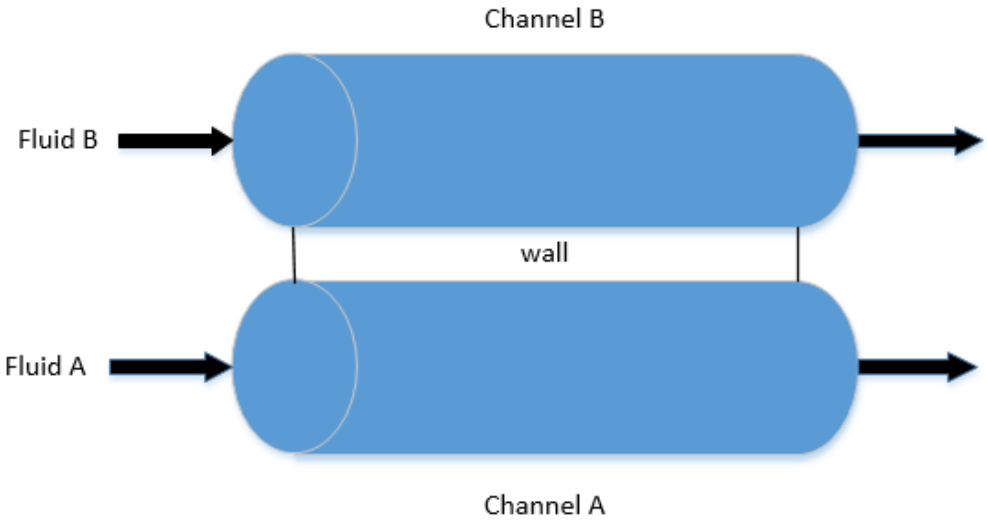

Figure 5: Cocurrent heat exchanger with fluid A and B flowing through the channels

2.2.1. Heat Exchanger by Equations

Heat exchanger mass balance equations for the fluids can be written as:

$$\frac{\partial \rho A}{\partial t} + \frac{\partial w}{\partial x} = 0$$

where ρ is the density of the fluid, A is the cross section and w is the mass flow rate. Density and the cross section were assumed to be constant for the fluids, hence mass flow rate was considered constant along the channels.

Heat exchanger energy balance equations were described considering a small portion $l = \frac{L}{N-1}$ on the channels where L is the length of each channel, and N is the number of nodes on the channels. Therefore, $N-1$ corresponds to the number of channel and wall segments. And, there are N-1 heat flow rates and N-1 temperature variables for the wall which are considered for the segments.

Discretized energy balance equations for the countercurrent heat exchanger was described as:

$$\rho_A l A_A c_{pA} \frac{dT_A(j+1)}{dt} + w_A c_{pA}[T_A(j+1) - T_A(j)] = Q_A(j)$$

$$\rho_B l A_B c_{pB} \frac{dT_B(N-j)}{dt} + w_B c_{pB}[T_B(N-j) - T_B(N-j+1)] = Q_B(N-j)$$

for $j = 1 \dots N-1$ where A_A and A_B are the cross section areas of the channels, ρ_A and ρ_B are the densities, c_{pA} and c_{pB} are the specific heat capacities, w_A and w_B are the mass flow rates of the fluids A and B respectively. Moreover, T_A and T_B are the temperature variables of the fluids A and B respectively, entering and exiting the small portion l. And, Q_A is the heat flow rate from wall to channel A, and Q_B is the heat flow rate from channel B to wall. In addition to this, boundary conditions were defined for the first node for T_A and for the node N for T_B.

The wall between the channels was assumed to be very thin, so its thermal resistance was neglected. Heat transfer occurs from the fluid A in channel A to the wall and from the wall to the fluid B in channel B. Energy balance at the each wall segment:

$$Q_A(j) = \gamma_A \omega_A l[T_w(j) - (T_A(j) + T_A(j+1))/2]$$

$$Q_B(N-j) = \gamma_B \omega_B[(T_B(N-j) + T_B(N-j+1))/2 - T_w(N-j)]$$

$$\frac{c_w}{N-1} \frac{dT_w(j)}{dt} = -Q_A(j) + Q_B(j)$$

for $j = 1 \dots N-1$ where γ_A and γ_B are the heat transfer coefficients of fluids A and B respectively, and ω_A and ω_B are the perimeter of the channels A and B respectively. Moreover, c_w is the specific heat capacity of the wall and T_w is the temperature variable of the wall segment.

Energy balance equations for cocurrent heat exchanger is very similar to the equations of countercurrent heat exchanger. The difference occurs because of the flow direction of the fluid B, therefore, in cocurrent heat exchanger boundary condition of T_B was described for the first node. And, boundary condition of T_A remained again for the first node.

Discretized energy balance equations for the cocurrent heat exchanger was described as:

$$\rho_A l A_A c_{pA} \frac{dT_A(j+1)}{dt} + w_A c_{pA}[T_A(j+1) - T_A(j)] = Q_A(j)$$

$$\rho_B l A_B c_{pB} \frac{dT_B(j+1)}{dt} + w_B c_{pB}[T_B(j+1) - T_B(j)] = -Q_B(j)$$

for $j = 1 \dots N-1$.

Energy balance at the each wall segment was modified in terms of temperature variables of fluid B:

$$Q_A(j) = \gamma_A \omega_A l[T_w(j) - (T_A(j) + T_A(j+1))/2]$$

$$Q_B(j) = \gamma_B \omega_B[(T_B(j) + T_B(j+1))/2 - T_w(j)]$$

$$\frac{c_w}{N-1}\frac{dT_w(j)}{dt} = -Q_A(j) + Q_B(j)$$

for $j = 1 \dots N-1$.

2.2.2. Steady State Analysis of Heat Exchanger

In Figure 6, temperature distributions of the heat exchangers in cocurrent and countercurrent mode are given. T_H and $T_c$represent the temperature of the hot fluid and the cold fluid, respectively.

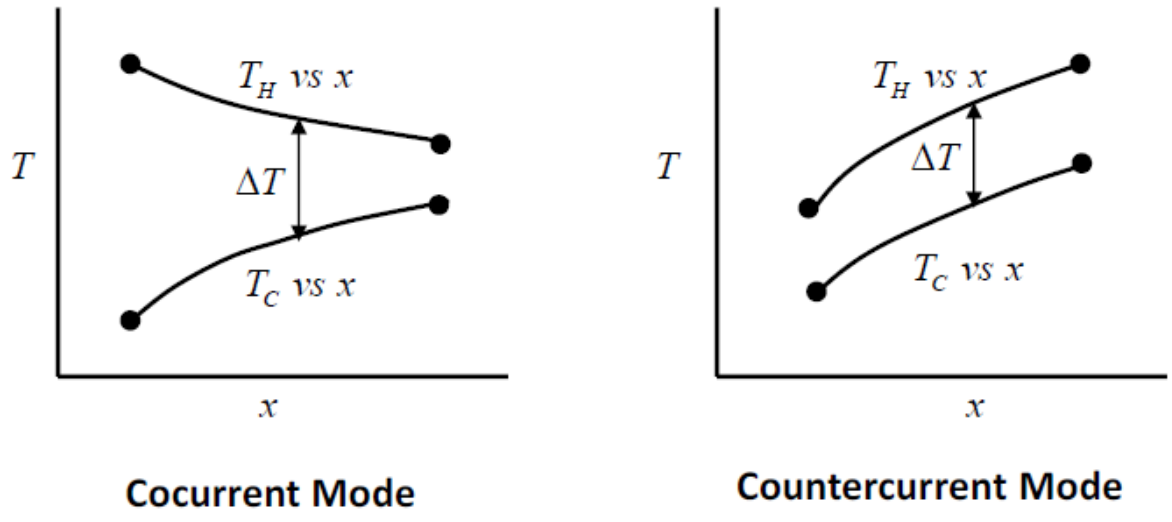

Figure 6: Cocurrent and countercurrent heat exchanger temperature distributions

ΔT represents the temperature difference between hot and cold fluids along the heat exchangers. And, x is a point along the channels of the heat exchanger. In the countercurrent

mode, ΔT does not vary along the channels as much as the ΔT in the cocurrent mode. Moreover, in cocurrent mode, ΔT is very large at the inlet of the channels and getting smaller progressively. Countercurrent heat exchanger can be evaluated as more efficient with respect to cocurrent heat exchanger since countercurrent mode requires smaller heat transfer area to provide the same heat transfer rate. The efficiency of countercurrent mode can also be seen from the plots in Chapter 3.

At the steady state, the total flow rates of Q_A and Q_B is equal to a steady state rate equation and it was used for the verification of the models.

Steady state heat rate equation for a heat exchanger is written as follows:

$$Q = UA\Delta T_{eog}$$

where U is the average overall heat transfer coefficient, A is the area of the heat transfer surface and ΔT_{eog} is the average temperature driving force.

UA is described as:

$$UA = L\omega \frac{\gamma_A \gamma_B}{\gamma_A + \gamma_B}$$

where L is the length, ω is the perimeter of the channels, γ_A and γ_B are the heat transfer coefficients of fluid A and B respectively.

ΔT_{eog} is written as:

$$\Delta T_{eog} = \frac{\Delta T_L - \Delta T_o}{\log(\frac{\Delta T_L}{\Delta T_o})}$$

where ΔT_L is the temperature difference of the fluids A and B at the outlet of the channels and $\Delta T_o$is the temperature difference of the fluids A and B at the inlet of the channels.

Therefore, steady state rate equations were described in OMEdit in order to verify the models.

2.3. Flexible Beam

As a flexible beam, a cantilever beam was modeled. Euler-Bernoulli theory was adopted to describe the beam bending which is considered to be acceptable for the thin beams. Thin beams imply that length of the beam is much greater than the height of it.

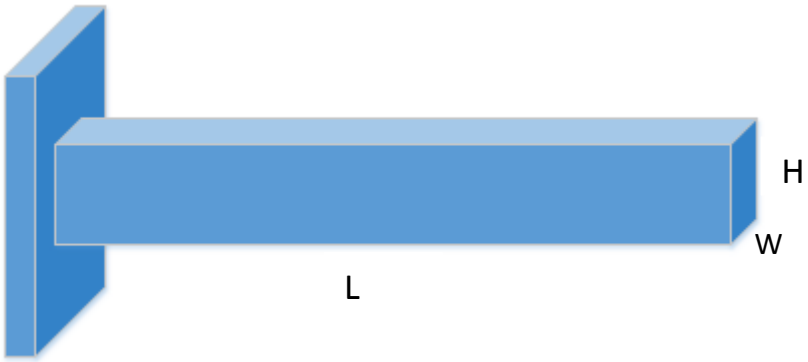

Figure 7: A cantilever beam

The flexible beam was implemented by using MSL and the model was verified by the analytical solution.

2.3.1. Flexible Beam by MultiBody Library

A flexible beam was created using Modelica.Mechanics.MultiBody library components. Flexible beam were approximated by the rigid bodies and joints coupled with springs and dampers. Flexible beam was discretized into N+1 body boxes and N revolute joints which provided the flexibility features to the model. And, N spring-damper components were placed to the revolute joints. The spring stiffness coefficients were determined depending on the material properties and the geometry of the flexible beam while damping coefficients were taken very small.

Modeling was adopted from the paper “Modeling Flexible Bodies in SimMechanics” by Victor Chudnovsky, Arnav Mukherjee, Jeff Wendlandt and Dallas Kennedy [8] which was intended for MATLAB and Simulink environment.

The beam was discretized into elements, and a single element was considered to be consisted of 2 body boxes and a revolute joint between these body boxes. If we assign length l to a single element, each body box in this element will have the length of $l/2$. By connecting each element together a flexible beam was obtained. Each element along the length of the beam was taken to be identical, therefore, the flexibility of the beam is uniform along its length.

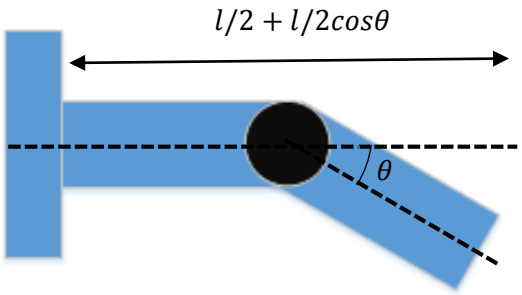

Figure 8: A single element of the flexible beam

Moreover, in order to provide flexibility to the beam, at each revolute joint a spring and a damper was placed. Determination of the spring stiffness came from the equality of the infinitesimal work done by the end of the single element and the joint. Moreover, the angle θ was considered to be very small and by the relation of the deflection, slope and moment, the spring constant in the form: $k = \frac{EI}{l}$ was obtained. In the equation, E is the modulus of elasticity, I is area moment of inertia and l is the length of a single element.

Therefore, the flexible beam was implemented depending on the characteristics as explained. At the fixed end of the flexible beam, the world component was placed. While using MultiBody library, it is mandatory to put the world component as it defines several features including gravity field for each component, as well as a fixed inertial frame of reference. In the model, zero gravity was selected and at the free end of the flexible beam a force component was placed in order to simulate the vibrations in the beam.

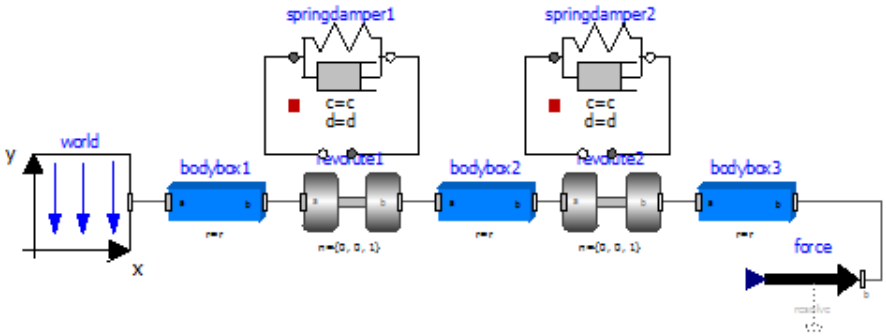

Figure 9: A flexible beam example containing 2 elements

2.3.2. Analytical Solution of Flexible Beam Model

The partial differential equation describing the motion of the infinitesimal element is:

$$EJ\frac{\partial^4 w}{\partial x^4} = -m\frac{\partial^2 w}{\partial t^2}$$

where m is the linear mass, w is the displacement, E is the young's modulus and J is the area moment of inertia.

Assuming a stationary solution of the bending motion in the form:

$$w(x,t) = \alpha(x)\beta(t)$$

where $\alpha(x)$ is a function of space alone describing the waveform of the stationary vibration and $\beta(t)$ is a time dependent vibration amplitude coefficient.

By placing the stationary solution $w(x,t)$ into the PDE and later defining:

$$\gamma^4 = \frac{m\omega^2}{EJ}$$

The natural frequencies and the modes of vibration of a beam in bending depend on physical parameters such as length, section, material and also boundary conditions. For the cantilever beam, there are four boundary conditions, two for the clamped end $x = 0$ and two for the free end $x = l$.

$$w(0,t) = 0 \qquad \left.\frac{\partial w}{\partial x}\right|_{x=0} = 0$$

$$\left.\frac{\partial^2 w}{\partial x^2}\right|_{x=l} = 0 \qquad \left.\frac{\partial^3 w}{\partial x^3}\right|_{x=l} = 0$$

The general solution becomes a linear combination of trigonometric equations:

$$\alpha(x) = A[cos(\gamma x) + cosh(\gamma x)] + B[cos(\gamma x) - cosh(\gamma x)] \\ + C[sin(\gamma x) + sinh(\gamma x)] + D[sin(\gamma x) - sinh(\gamma x)]$$

and after placing the boundary conditions into the function $\alpha(x)$, we reduce the function into a new form. Moreover, from the boundary conditions it is obtained: $A = C = 0$ and;

$$D = B\frac{-cos(\gamma L) - cosh(\gamma L)}{sin(\gamma L) + sinh(\gamma L)}$$

Thus, it can be written in the form:

$$\alpha(x) = B\{[cos(\gamma x) - cosh(\gamma x)] + \left[\frac{-cos(\gamma L) - cosh(\gamma L)}{sin(\gamma L) - sinh(\gamma L)}\right][sin(\gamma x) - sinh(\gamma x)]\}$$

where $B = 1/2$. Moreover, by the boundary conditions, we obtain the frequency equation for the cantilever beam: $\cosh(\gamma l)\cos(\gamma l) = -1$. We have infinite number of solutions for γ_k , $k = 1,2, \ldots$. And, we obtain the natural frequencies by using the equation:

$$\omega_k = (\gamma_k)^2\sqrt{\frac{EJ}{m}}$$

By solving the frequency equation we obtained $\gamma_k l$ and ω_k, values for the first 6 modes are given in the Table 1:

Table 1: Values for the first 6 modes

k	γ_k	ω_k	$Frequency(Hz)$
1	1,875	410,4352	65,3228
2	4,694	2,5723e3	409,4005
3	7,855	7,2033e3	1,1464e3
4	10,9955	1,4115e4	2,2464e3
5	14,1371	2,3333e4	3,7135e3
6	17,2787	3,4855e4	5,5473e3

Therefore, for each frequency, there is a characteristic vibration:

$$w_k(x,t) = \alpha_k(x)\beta_k(t) = \alpha_k(x)[A_k \cos(\omega_k t)]$$

An approximation for A_k is given below according to P which is force. It is calculated as if the beam starts vibration at t=0.

$$A_k = \frac{-4PL}{EJm{\gamma_k}^4(\sin(\gamma_k L)\, e^{\gamma_k L} + e^{2\gamma_k L} - 1)} * [3\sin(\gamma_k L)\,(e^{2\gamma_k} + 1) - 2(\gamma_k L)^3 e^{\gamma_k L} + \cos(\gamma_k L)(3 - (\gamma_k L)^3(e^{2\gamma_k L} + 1) - 3e^{2\gamma_k L})]$$

Therefore, this solution was used to verify the flexible beam implemented by MultiBody library and the results are provided in Chapter 3.

2.4. String

A string was modeled by considering the behavior of a travelling wave on a string. The prescribed transversal displacement at one end creates a wave on the string. From one end of the string model, a pulse was given and a travelling wave was observed along the string and it was modeled by using MultiBody library.

2.4.1. String by MultiBody library

In order to be able to observe the characteristics of a string, a pulse was given on the horizontal axis and while there was a uniform gravity on the vertical axis, movement of the pulse was observed along the created model. The main idea was to see a movement of the rigid bodies such that reflects a travelling wave when a pulse was given from one end of the string model.

The main body of the string was created by body boxes and revolute joints coupled with dampers. The travelling wave is determined by the coupling between inertia and tension due to gravity. Moreover, by using necessary components of MSL a pulse was applied to the string.

With the world component gravity was applied on the vertical axis. The prismatic joint was coupled with a position component which provided a horizontal movement to the prismatic joint. And, position component was controlled with a pulse input which had 1 period.

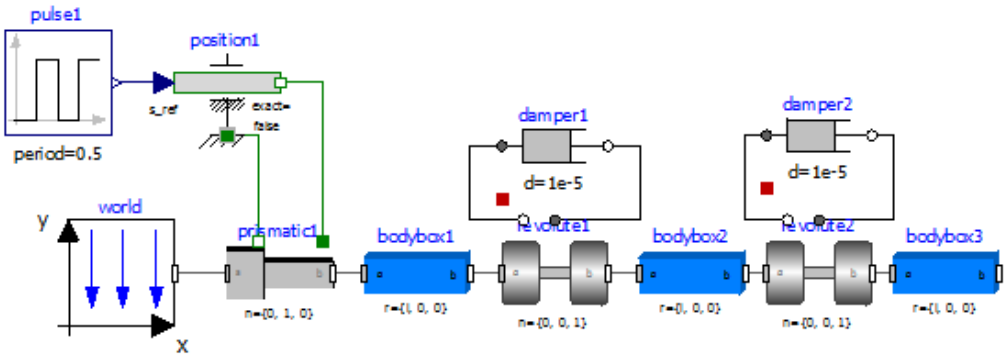

Figure 10: A string model example

An example of a string model is given in Figure 10 when N=2 where N corresponds to the number of revolute joints in the model, therefore, there are N+1 body boxes. Position component enables one to filter the input signal in order to eliminate the high frequency components. Hence, in our model pulse signal was filtered by the position component in order to provide a slow pulse to the string and by that way the response of the body boxes was observed.

Parameters were defined using Modelica.SIunits library. In Chapter3, plots and statistics for increasing N values are provided.

2.5. Transmission Line

A transmission line is used to carry an electrical signal from one point to another. Transmission lines are widely used in our daily lives. They can be used for many purposes such as connecting a transmitter or a receiver to an antenna, connecting the computers in a network and so on.

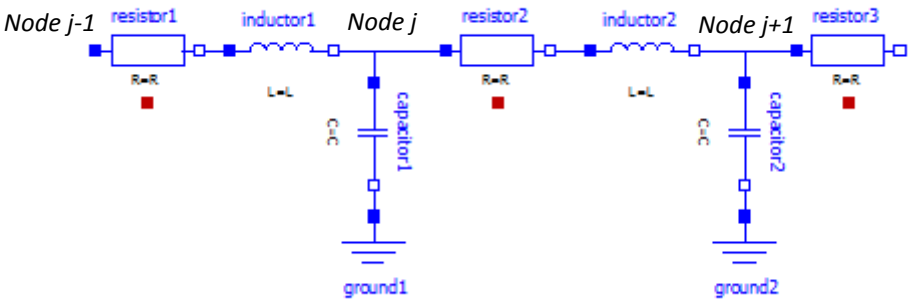

Figure 11: Transmission Line example

In Figure 11, it is given an example of the transmission line that was implemented which consists of a resistor, an inductor and a capacitor within each segment. Moreover, the transmission line was implemented by connecting each segment together. In the figure, resistor1, inductor1 and capacitor1 describes the first segment and it is connected to second segment which has the same components as the first segment and so on. It transmits the electrical signal from a source to a load. Furthermore, the first segment will be connected to a voltage source or a filter which can be used eliminate the high frequency oscillations, and the last segment is connected to a load.

In our case, a second order low pass filter was placed between a step voltage and the transmission line, and as a load, a matched load resistance was chosen.

Transmission line was implemented both using Modelica.Electrical library and equations.

2.5.1. Transmission Line Circuit by Electrical library

The transmission line was implemented as it is explained in the section 2.5. Its parameters are N “number of segments”, r “resistance per meter”, l “inductance per meter”, c “capacitance per meter” and L “length of the transmission line”.

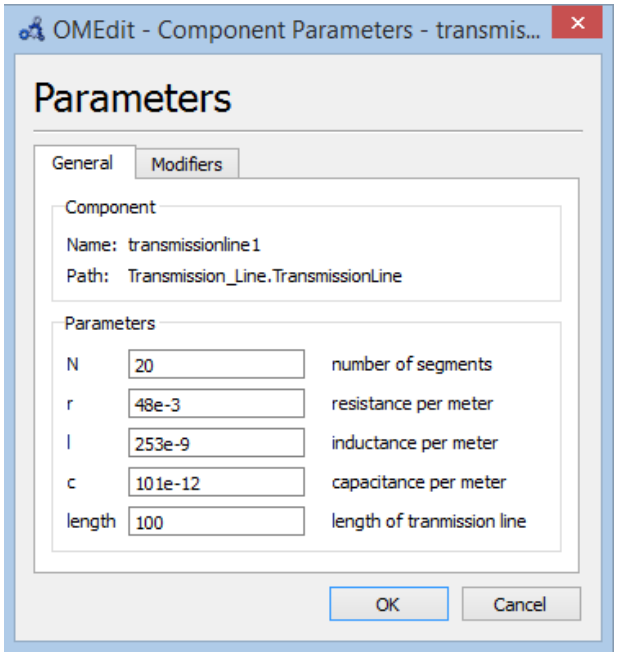

Figure 12: Transmission Line parameters window

Two pins were placed at the ends of the transmission line. Left pin was connected to the resistor of the first segment and the right pin was connected to the inductor of the last segment.

Figure 13: Transmission Line icon

To the left pin of the transmission line, controlled voltage component was connected which is a parameterless converter of real valued signals into a source voltage. Moreover, the real valued signals had to be provided by the components of Modelica.Blocks library. A step input was selected and it was filtered with a second order low pass filter in order to eliminate its high frequency components. And, at the right end of the transmission line, a load resistance was placed which was equal to the characteristic impedance of the transmission line.

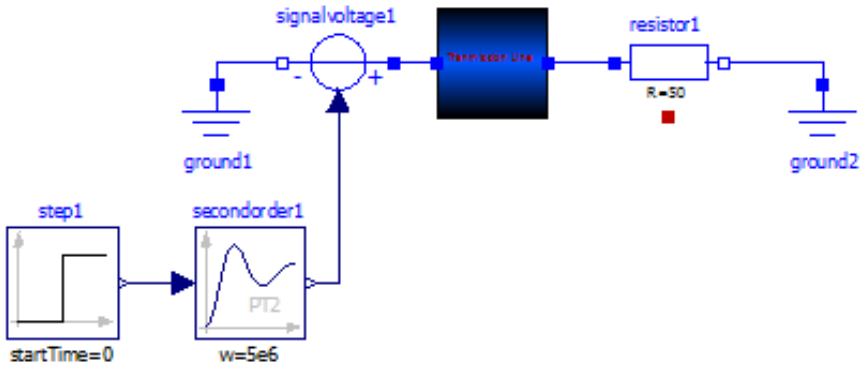

Figure 14: Transmission Line circuit model

If the distributed resistance is neglected, characteristic impedance of the transmission line can be approximated by:

$$Z_o = \sqrt{\frac{l}{c}}$$

where l and c is the inductance per meter and capacitance per meter respectively.

In the transmission line circuit model, load resistance was taken as equal to the characteristic impedance in order to test the matched load case in which no reflected wave is observed in the transmission line.

Another important point of the transmission line that has to be analyzed is the time delay between its ends. It can be calculated as:

$$\tau = \sqrt{\frac{L}{v}}$$

where L is the length of the transmission line and $v = \frac{1}{\sqrt{lc}}$ is the velocity of the signal. The formula of the time delay was used for the verification of the implemented transmission line circuit model to check whether the time delay of the model satisfies the theoretical one.

The second order low pass filter was in the form:

$$y = \frac{k}{(\frac{s}{\omega})^2 + 2D\frac{s}{\omega} + 1}u$$

where ω is the angular cut-off frequency, D is the damping, k is the gain, u is the step input and y is the output which is the input for signal voltage component. Second order filter provided a slope of -40dB/decade which mostly removed undesired high frequency oscillations. The angular cut-off frequency was determined by measuring the high frequency oscillations as well as paying attention not to remove the desired frequencies. Moreover, damping and gain of the filter was chosen to be 1.

2.5.2. Transmission Line Circuit by Equations

Considering the nodes of the discrete transmission line shown in Figure 11, circuit equations can be written. In the transmission line, we considered N segments, therefore, there were N+1 nodes, and N+1 voltage and current variables.

$$C_x \frac{dv_{j+1}}{dt} = \frac{i_j - i_{j+1}}{l}$$

$$L_x \frac{di_j}{dt} = -R_x i_j + \frac{v_j - v_{j+1}}{l}$$

where $j = 2, .., N$. And, $l = \frac{L}{N}$ is the length of each segment, $R_x$is the resistance per meter, L_x is the inductance per meter and C_x is the capacitance per meter.

Output voltage can be described as:

$$v_{N+1} = i_{N+1} R_L$$

where R_L is the load resistance which was taken to be equal to the characteristic impedance of the transmission line.

Moreover, considering the form of the second order low pass filter as mentioned in Section 2.5.1, equation of the filter can be defined in the following way:

$$V_{step} = v_1 + \frac{2}{\omega}\frac{dv_1}{dt} + \frac{1}{\omega^2}\frac{d^2v_1}{dt^2}$$

where V_{step} is the step voltage and v_1 is the output voltage of the filter.

The equations were defined in OMEdit as explained. And, parameters and variables were defined using Modelica.SIunits library. The results of the transmission line circuit models are discussed in Chapter 3.

3. Simulations

The implemented models were simulated and evaluated in terms of the plots of the results, compilation times and simulation times. Plots and time spent for the compilation and simulation of the models are important in order to analyze the performance of Modelica while handling large systems which means high number of equations. For the purpose of the performance and plot analysis, each model was simulated by gradually increasing "N" of the models. The test system was a laptop with AMD Quad-Core Processor @ 1.4 GHz and 4 GB RAM running Windows 8. All the models were simulated using the dassl-solver for a time period necessary to observe the expected results.

OpenModelica compilation works by translating a Modelica code to a C code to make it executable for the simulation. Modelica source code is first translated to a flat model, which means in this phase type checking, inheritance, generation of connection equations are performed. Later, in the analyzer and optimizer, sorting of the equations and optimization of equations are done which are necessary for compiling the models. Finally, sequential C code is generated and compiled to produce executable code.

In this chapter, statistics of each model is provided within the tables in terms of compilation and simulation times. For the compilation time, specifically, timeSimCode and timeCompile, which are the time for generating C code from optimized sorted equations and the time of C code compilation respectively, are highlighted.

Simulations were implemented using command shell by creating a .mos file which is a Modelica script file. Furthermore, statistics of the compilation and simulation time were written in a .txt file in which time spent for compilation and simulation can be seen. Moreover, the generated code and the plots of the results were obtained.

An example is given below about how to create a .mos file by using Visual Figaro:

TransmissionLine.mos:

loadModel(Modelica);
loadFile("ScalableTestSuite.mo"); getErrorString();
simulate(ScalableTestSuite.TransmissionLine.TransmissionLineEquations_N_2, simflags = "-lv LOG_STATS"); getErrorString();

After creating the .mos files, OMC was invoked using command shell as shown in Figure 15.

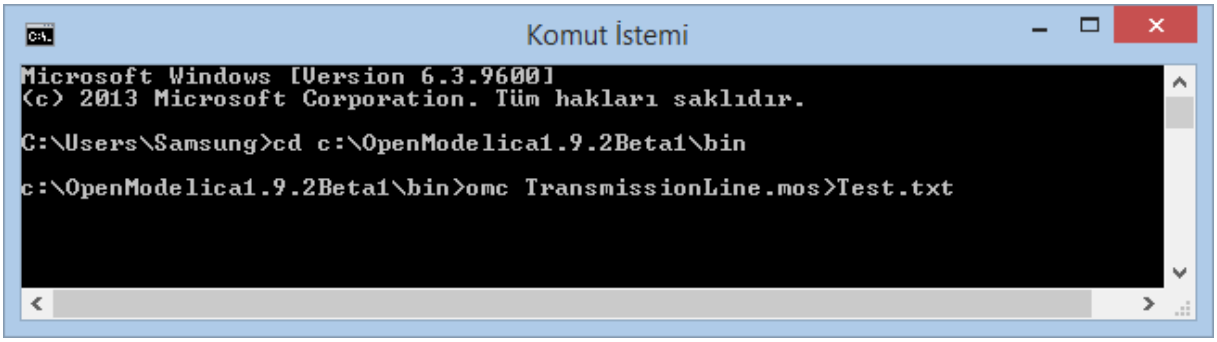

Figure 15: Command shell OMC invocation

In the example provided in Figure 15 statistics are written in "Test.txt" file which contains compilation and simulation times. Generated code is created in a file named "TransmissionLineEquations_N_2.exe" and the plots of the results are provided in a file named "TransmissionLineEquations_N_2.mat".

Therefore, all the models were simulated in same way and the results are discussed.

3.1. Simulations of Heat Conduction Models

For the heat conduction models, as mentioned in Chapter 2 there were two kind of models which are mentioned as HeatConductionTT and HeatConductionTI.

An aluminum material was considered for the simulations. Material parameters are shown in Table 2.

Table 2: Parameters for heat conduction simulations

Length(m)	Specific Heat Capacity (J/(kg.K))	Thermal conductivity (W/(m.K))	Density (kg/m^3)
0.2	910	237	2712

Both models were implemented using their equations and Modelica.Thermal library. Furthermore, verification was done by the analytical solutions of the models which were calculated according to their boundary conditions.

Mentioning again that N is the number of nodes for the heat conduction models, 32 simulations were done in total for different N values. 16 simulations were done for both HeatConductionTI and HeatConductionTT: 8 for the models implemented by equations and 8 for the models implemented by Modelica.Thermal library.

Boundary conditions and N values for HeatConductionTT and HeatConductionTI which were used in the simulations are shown in Table 3, 4 and 5.

Table 3: HeatConduction TT boundary conditions and simulation time

Temperature at the first node(N=1)	Temperature at the last node (node N)	Initial condition for the other nodes(2,..,N-1)	Simulation Time
330 K	300 K	273.15 K	350 s

Table 4: HeatConductionTI boundary conditions and simulation time

Temperature at the last node (node N)	Initial condition for the other nodes(1,..,N-1)	Simulation Time
330 K	273.15 K	1500 s

Table 5: N values for the heat conduction simulations

N (number of nodes)
10
20
40
80
160
320
640
1280

3.1.1. Plots of HeatConductionTT and HeatConductionTI

For the verification of HeatConductionTT and HeatConductionTI, plots when N=1280 are shown since they are the most accurate.

In the verification, mid-nodes of the models were selected, and verified by the analytical solutions as seen in Figure 16 and 17. In the figures, red lines represent the temperature at the middle of the models implemented by analytical solutions, blue lines represent the temperature at the mid-node of the models implemented by equations and green lines represent the temperature of the heat capacitor at the middle of the models implemented by Modelica.Thermal library. According to Figure 16 and 17, results of the equations and Thermal library are exactly matching with each other and with the analytical solutions.

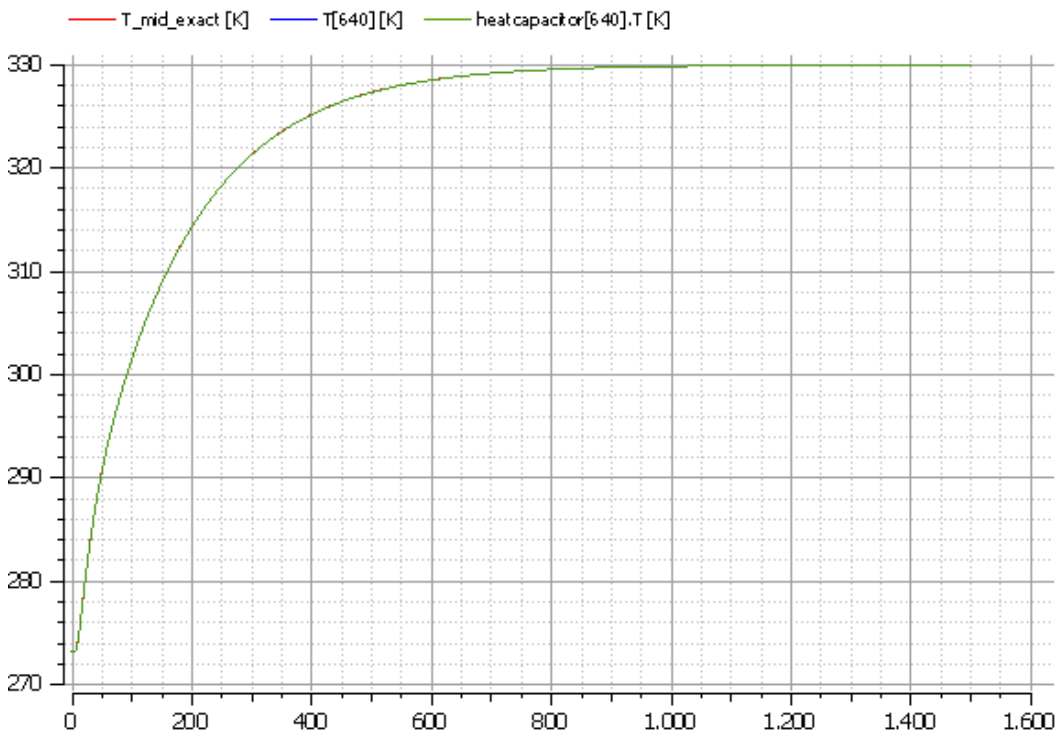

Figure 16: Verification for HeatConductionTI when N=1280

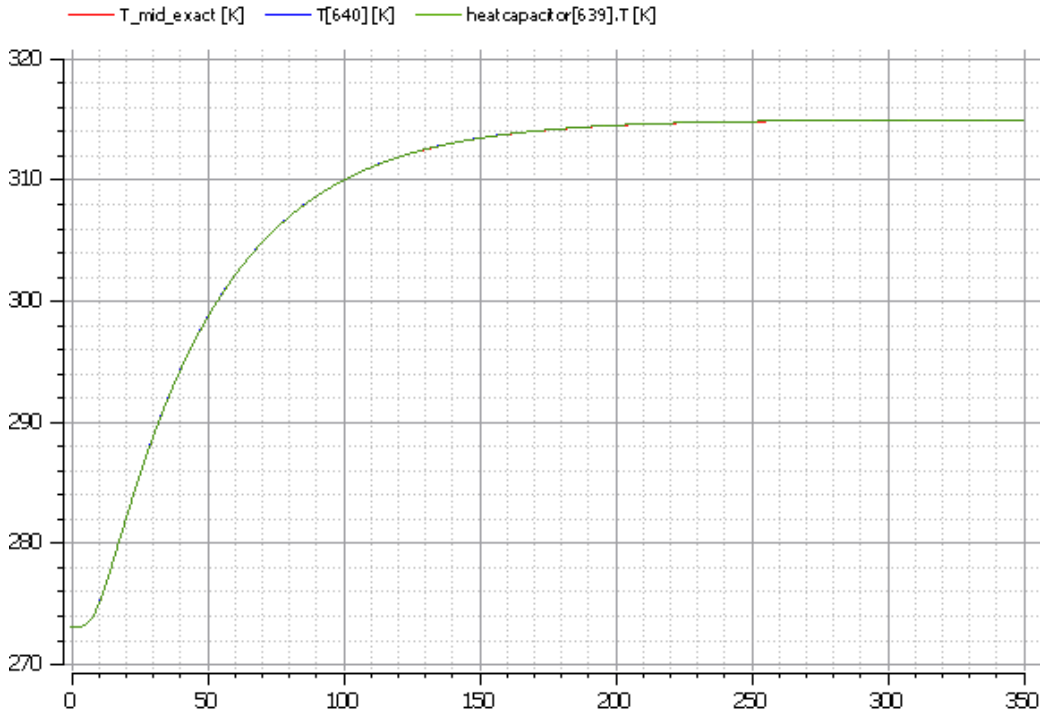

Figure 17: Verification for HeatConductionTT when N=1280

In the next two figures, some representative temperatures for HeatConductionTI models with respect to time are given for our N values which are shown in Table 5. In Figure 18, lines represent the temperatures at the mid-nodes of each HeatConductionTI model implemented by equations. And, in Figure19, lines represent the temperatures of the heat capacitors at the middle of each HeatConductionTI model implemented by MSL.

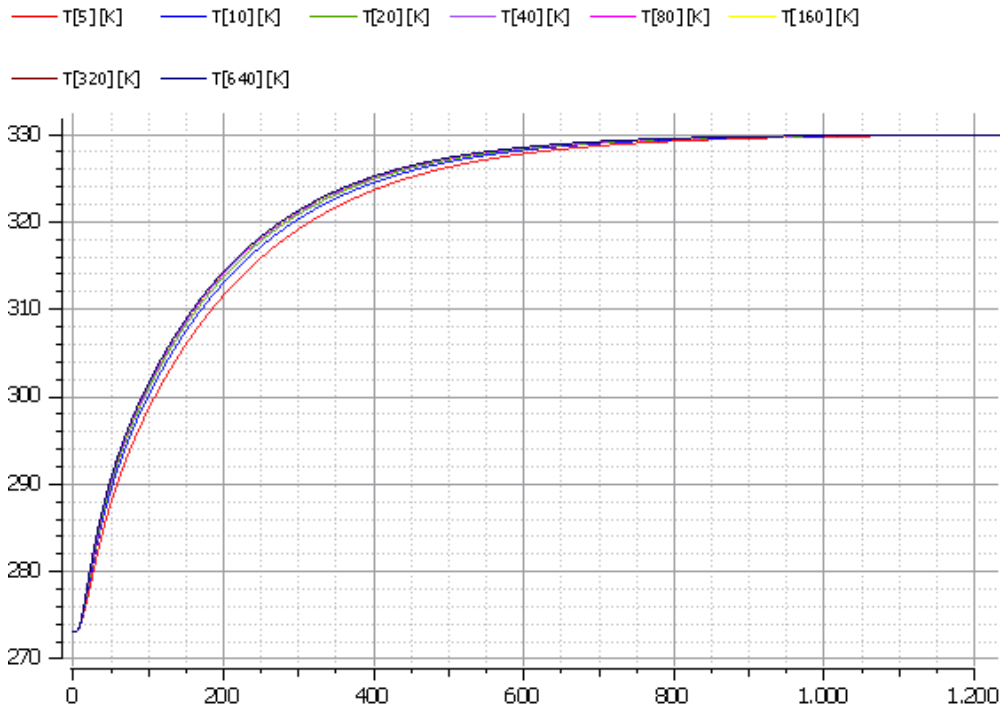

Figure 18: Temperatures of HeatConductionTI implemented by equations

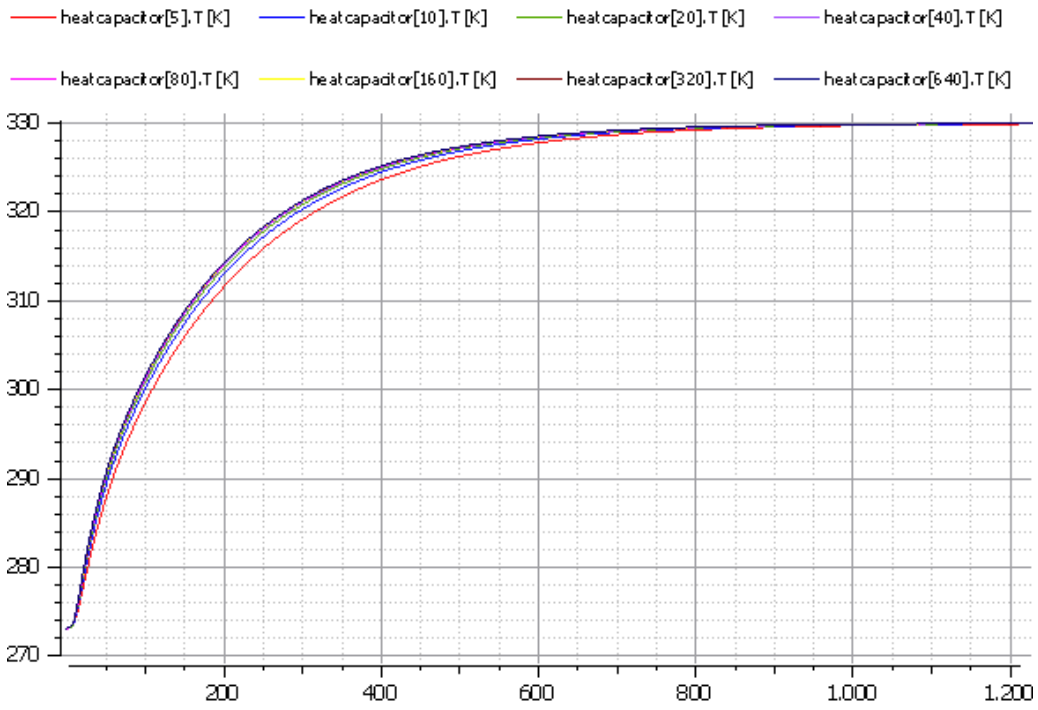

Figure 19: Temperatures of HeatConductionTI implemented by MSL

The temperatures of HeatConductionTI implemented by equations and MSL are exactly matching at each N and they give the same response with respect to time. Temperatures are starting from 273.15 K which is the initial temperature and they reach steady state value 330K around 1200 seconds. In Figure 18 and 19, red lines represent the temperatures when N=10, and as N increases error decreases between the expected temperature. Starting from N=320, temperatures show the same response for the increasing N values.

In Figure 20 and 21, the necessary temperatures for HeatConductionTT models with respect to time are given for the same N values. In Figure 20, lines represent the mid-node temperatures of the HeatConductionTT model implemented by equations.

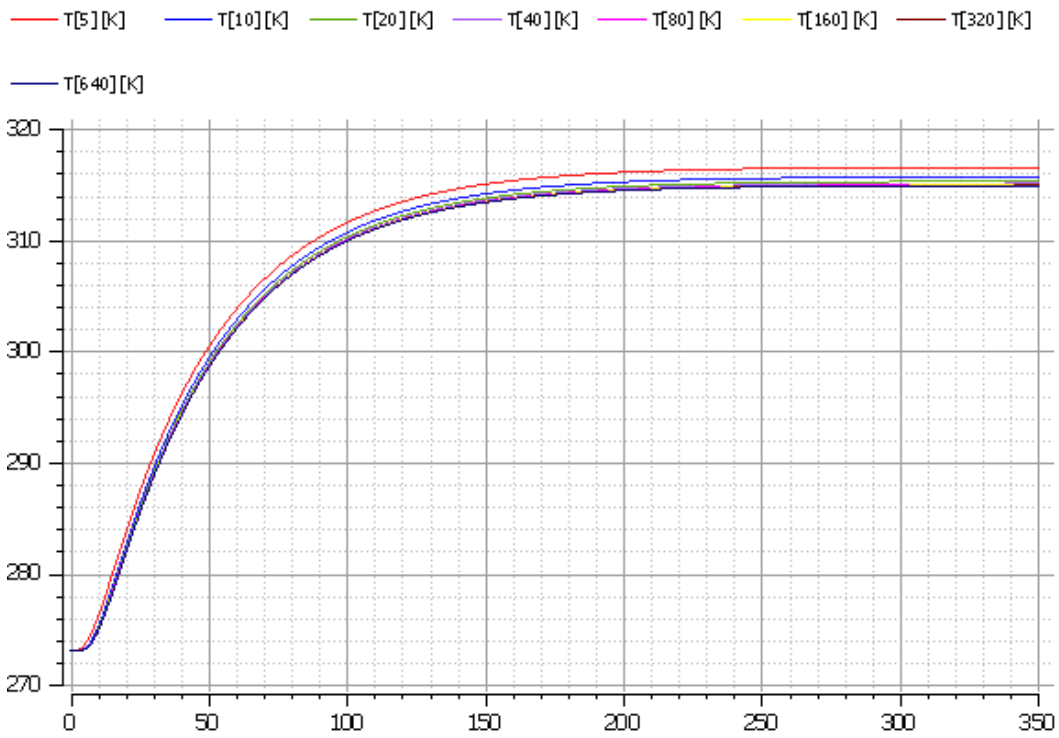

Figure 20: Temperatures of HeatConductionTT implemented by equations

In Figure 21, lines represent the temperatures of the heat capacitors at the middle of the HeatConductionTT models implemented by MSL.

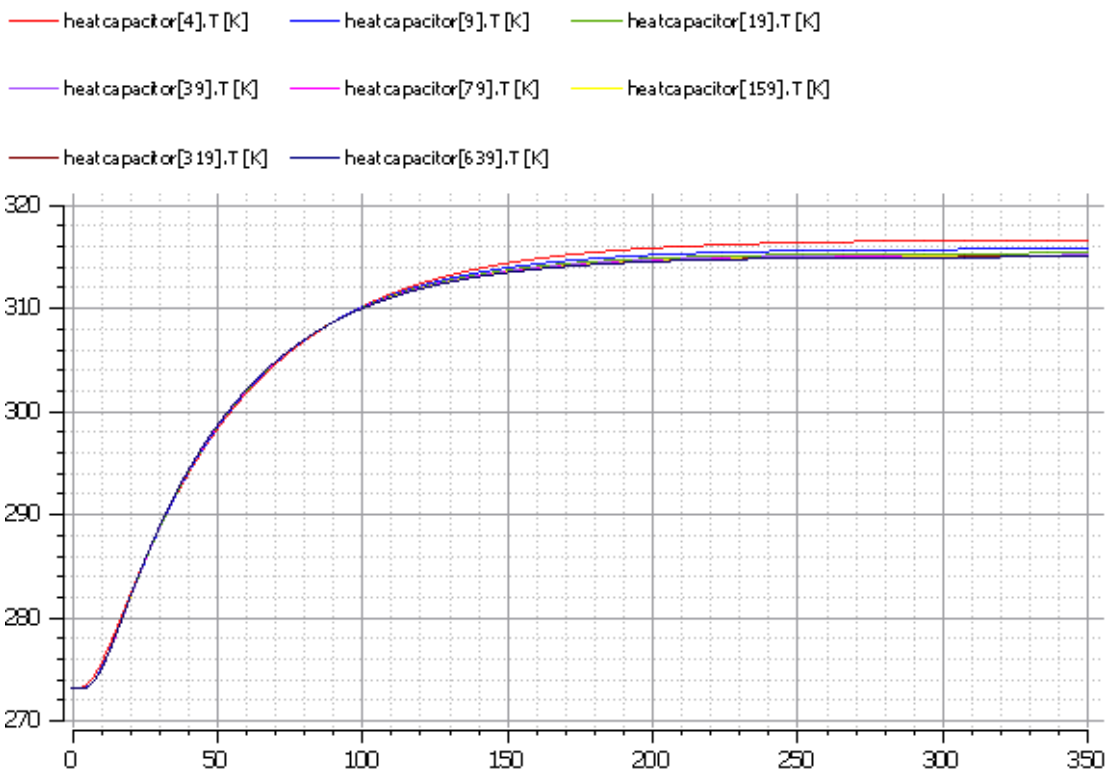

Figure 21: Temperatures of HeatConductionTT implemented by MSL

In both models of HeatConductionTT, mid-node temperatures are reaching steady state temperature 315K around 300 seconds. While N=10 and N=20 the temperature differences between the models implemented by equation and MSL are apparent, as N increases this difference is getting lost.

The plots of HeatConductionTT and HeatConductionTI were as expected. When a rod is exposed to fixed temperatures from its both ends it reaches steady state faster than the rod which is exposed to a fixed temperature from one end. In the plots of HeatConductionTI, it is seen that equations and MSL show exactly the same results. However, in HeatConductionTT, plots proves that MSL shows better results than equations at lower N values as can be seen from the error between N=10 and N=1280 in the Figure 20 and 21.

3.1.2. Statistics of HeatConductionTT and HeatConductionTI

In this part, compilation and simulation times of the models are provided. Statistics for HeatConductionTT implemented by equations are given in Table 6.

Table 6: Statistics for HeatConductionTT implemented by equations

N=10	**Compilation Time**	12,7201 s	**timeSimCode**	0,059 s
			timeCompile	10,5358 s
	Simulation Time	0,02205 s		
N=20	**Compilation Time**	12,6075 s	**timeSimCode**	0,1237 s
			timeCompile	9,9512 s
	Simulation Time	0,0256 s		
N=40	**Compilation Time**	13,5012 s	**timeSimCode**	0,2632 s
			timeCompile	10,3084 s
	Simulation Time	0,0449 s		
N=80	**Compilation Time**	14,7156 s	**timeSimCode**	0,5387 s
			timeCompile	10,8068 s
	Simulation Time	0,0888 s		
N=160	**Compilation Time**	18,5574 s	**timeSimCode**	1,7197 s
			timeCompile	12,1735 s
	Simulation Time	0,2054 s		
N=320	**Compilation Time**	26,4369 s	**timeSimCode**	2,5133 s
			timeCompile	15,3876 s
	Simulation Time	0,4899 s		
N=640	**Compilation Time**	42,0018 s	**timeSimCode**	5,6619 s
			timeCompile	19,7147 s
	Simulation Time	1,6681 s		
N=1280	**Compilation Time**	85,6352 s	**timeSimCode**	15,6075 s
			timeCompile	30,6983 s
	Simulation Time	6,1233 s		

Statistics for HeatConductionTT implemented by MSL are given in Table 7.

Table 7: Statistics for HeatConductionTT implemented by MSL

N=10	**Compilation Time**	13,3162s	**timeSimCode**	0,047 s
			timeCompile	10,8379s
	Simulation Time	0,02727 s		
N=20	**Compilation Time**	13,5027 s	**timeSimCode**	0,1013 s
			timeCompile	10,2659 s
	Simulation Time	0,0498 s		
N=40	**Compilation Time**	15,6125 s	**timeSimCode**	0,2394 s
			timeCompile	10,6928 s
	Simulation Time	0,0961 s		
N=80	**Compilation Time**	19,2601 s	**timeSimCode**	0,5348 s
			timeCompile	12,2513 s
	Simulation Time	0,196 s		
N=160	**Compilation Time**	26,6549 s	**timeSimCode**	1,1833 s
			timeCompile	14,2433 s
	Simulation Time	0,4042 s		
N=320	**Compilation Time**	44,1781 s	**timeSimCode**	3,7143 s
			timeCompile	19,6767 s
	Simulation Time	0,9081 s		
N=640	**Compilation Time**	105,5254 s	**timeSimCode**	11,4071 s
			timeCompile	30,2928 s
	Simulation Time	2,6541 s		
N=1280	**Compilation Time**	299,5425 s	**timeSimCode**	36,9461 s
			timeCompile	52,2366 s
	Simulation Time	8,1803 s		

Figure 22 compares the compilation times of HeatConductionTT implemented by equations and MSL according to our N values.

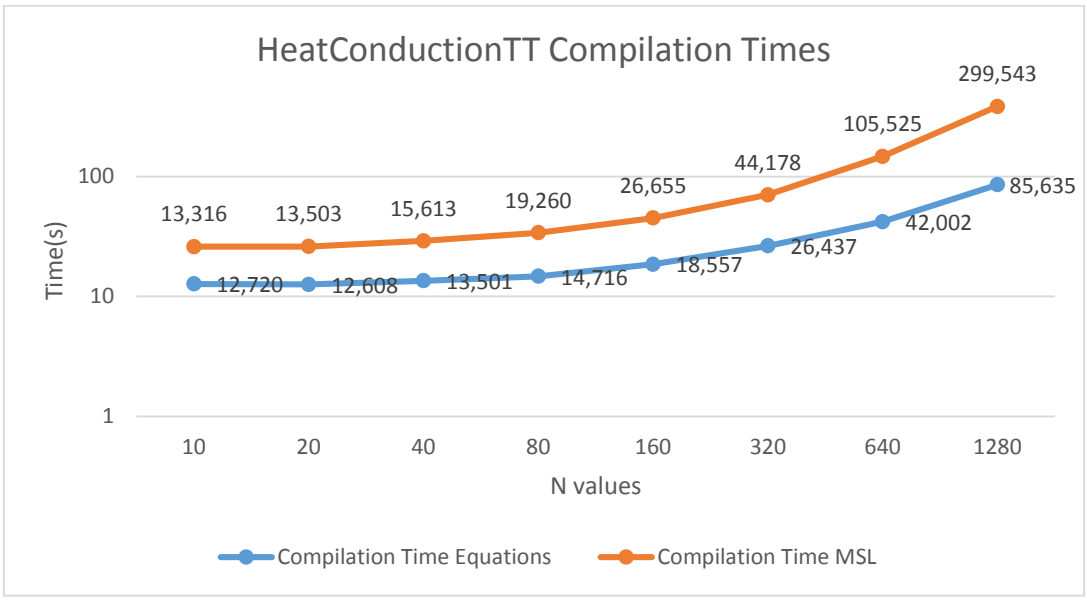

Figure 22: HeatConductionTT compilation times

Compilation times starts increasing after N=20 for the HeatConductionTT models, and time difference between the compilation times of the models implemented by equations and MSL is growing as N increases. For large values of N, compilation times of the model implemented by MSL increases roughly with a second power of N. And, for low values of N, the time is almost constant due to a fixed overhead independent of size.

In Figure 23, simulation times for HeatConductionTT implemented by equations and MSL are given according to our N values.

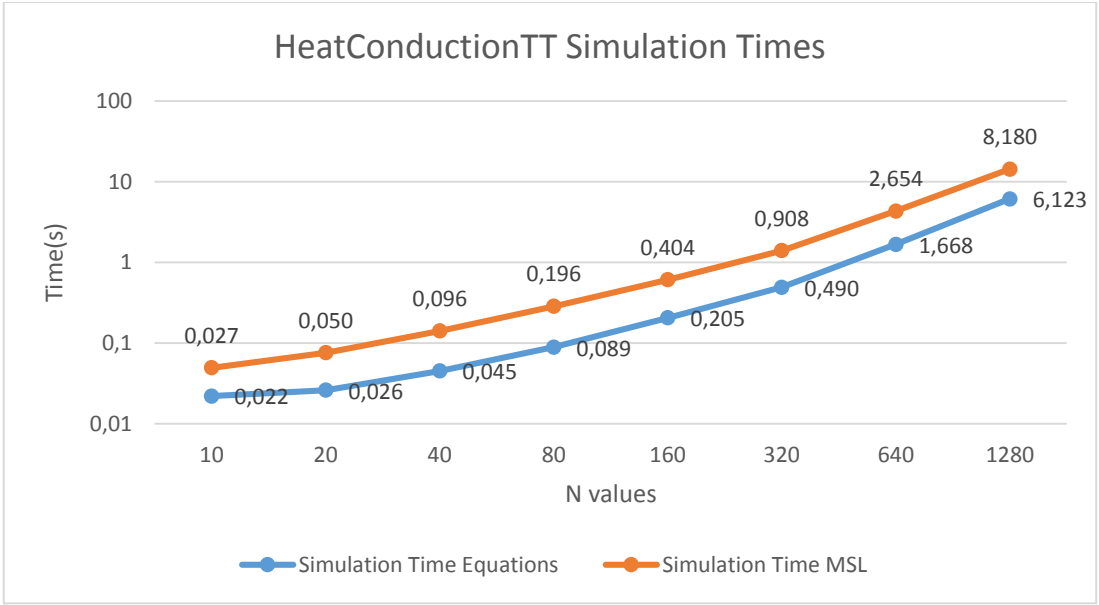

Figure 23: HeatConductionTT simulation times

As it can be inspected in Figure 23, simulation times of HeatConductionTT models increase with second power of N.

Statistics for HeatConductionTI implemented by equations are given in Table 8.

Table 8: Statistics for HeatConductionTI implemented by equations

N=10	**Compilation Time**	12,4957 s	**timeSimCode**	0,0649 s
			timeCompile	10,1706 s
	Simulation Time	0,0154 s		
N=20	**Compilation Time**	12,1556 s	**timeSimCode**	0,1305 s
			timeCompile	10,1645 s
	Simulation Time	0,02511 s		
N=40	**Compilation Time**	13,1739 s	**timeSimCode**	0,2637 s
			timeCompile	10,3102 s
	Simulation Time	0,0492 s		
N=80	**Compilation Time**	14,7031 s	**timeSimCode**	0,5516 s
			timeCompile	10,7217 s
	Simulation Time	0,0918 s		
N=160	**Compilation Time**	18,4264 s	**timeSimCode**	1,1664 s
			timeCompile	12,6475 s
	Simulation Time	0,205 s		
N=320	**Compilation Time**	25,8326 s	**timeSimCode**	2,5179 s
			timeCompile	14,7186 s
	Simulation Time	0,5185 s		
N=640	**Compilation Time**	42,9075 s	**timeSimCode**	6,4962 s
			timeCompile	19,4988 s
	Simulation Time	1,7475 s		
N=1280	**Compilation Time**	82,2501 s	**timeSimCode**	14,6987 s
			timeCompile	30,2779 s
	Simulation Time	5,968 s		

Statistics for HeatConductionTI implemented by MSL are given in Table 9.

Table 9: Statistics for HeatConductionTI implemented by MSL

N=10	**Compilation Time**	13,0588 s	**timeSimCode**	0,0495 s
			timeCompile	10,4661 s
	Simulation Time	0,027 s		
N=20	**Compilation Time**	13,1929 s	**timeSimCode**	0,1101 s
			timeCompile	10,3077 s
	Simulation Time	0,0514 s		
N=40	**Compilation Time**	14,3168 s	**timeSimCode**	0,2232 s
			timeCompile	10,7568 s
	Simulation Time	0,0988 s		
N=80	**Compilation Time**	17,5525 s	**timeSimCode**	0,4961 s
			timeCompile	12,1652 s
	Simulation Time	0,1931 s		
N=160	**Compilation Time**	24,2966 s	**timeSimCode**	1,1636 s
			timeCompile	13,9298 s
	Simulation Time	0,4012 s		
N=320	**Compilation Time**	42,5138 s	**timeSimCode**	3,0872 s
			timeCompile	19,5183 s
	Simulation Time	0,9692 s		
N=640	**Compilation Time**	102,8176 s	**timeSimCode**	9,7597 s
			timeCompile	30,4896 s
	Simulation Time	2,7141 s		
N=1280	**Compilation Time**	298,0852 s	**timeSimCode**	37,887 s
			timeCompile	52,0476 s
	Simulation Time	8,135 s		

In Figure 24, the compilation times of HeatConductionTI implemented by equations and MSL are shown according to our N values.

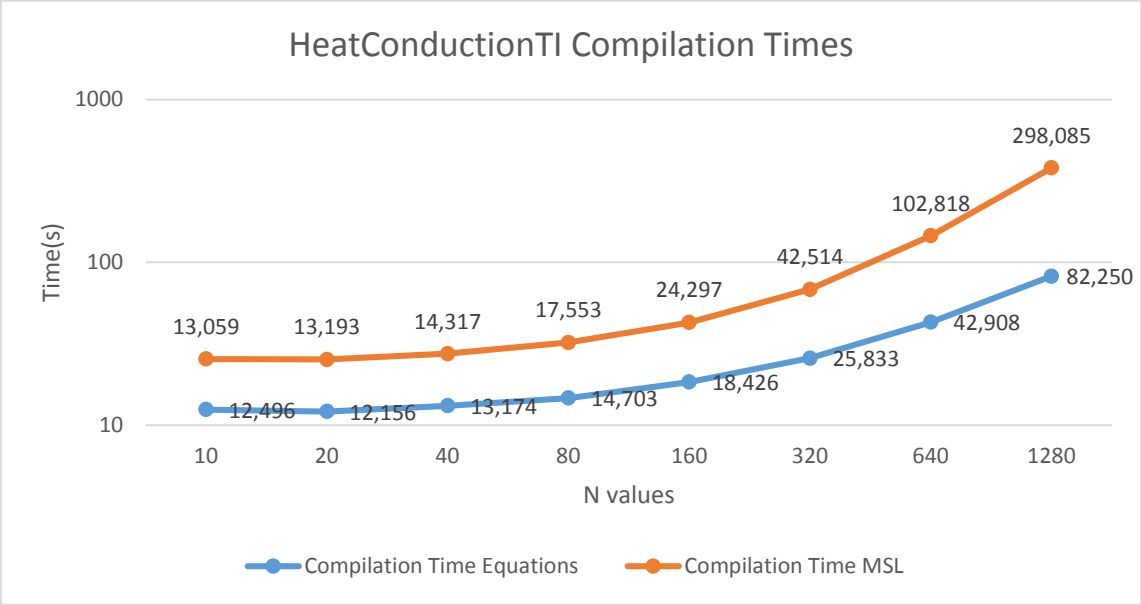

Figure 24: HeatConductionTI compilation times

Compilation and simulation times of HeatConductionTI models are very near to HeatConductionTT models.

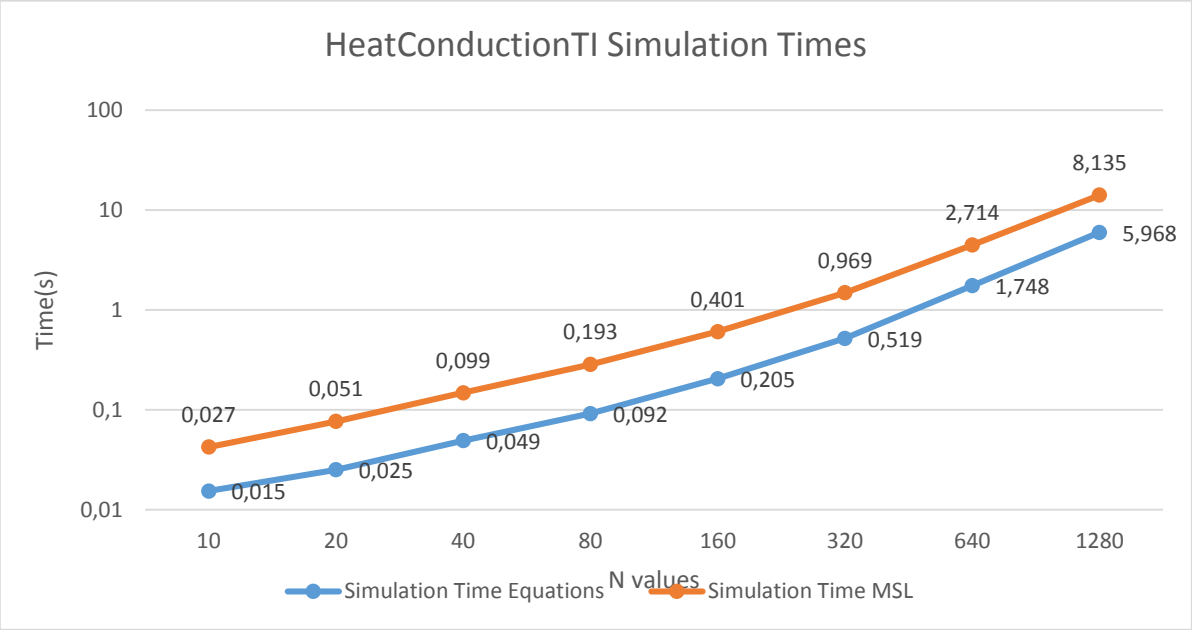

Figure 25: HeatConductionTI simulation times

It is observed that HeatConductionTI and HeatConductionTT models show the same compilation and simulation times. However, the models implemented by Thermal library take more time than the models implemented by equations in terms of compilation and simulation.

3.2. Simulations of Heat Exchanger Models

As it is mentioned in Chapter 2, there were two models of heat exchangers implemented by equations; countercurrent heat exchanger and cocurrent heat exchanger. Heat exchangers were simulated for the same parameter values. Their difference occurred due to boundary conditions which was caused by the flow direction of fluid B.

Parameters used for both fluid A and B for both heat exchangers:

Table 10: Parameters of fluid A and B

Density (kg/m^3)	Specific heat capacity (J/(kg.K))	Heat Transfer Coefficient ($W/(m^2.K)$)
1000	4200	4000

Parameters used for channel A and B and wall for both heat exchangers:

Table 11: Parameters of channel A and B

Length (m)	Cross sectional area (m^2)	Perimeter (m)	Specific heat capacity of the wall (J/(kg.K))
10	$5*10^{-5}$	0.1	2000

Mass flow rates of fluid A and B for both heat exchangers:

Table 12: Mass flow rates of fluid A and B

Mass flow rate of B (kg/s)	Mass flow rate of A (kg/s)
1	if time<15 seconds then 1 else 1.1

As mentioned in Chapter 2, in the heat exchangers, there are N temperature variables on the channels. And, N-1 heat flow rate variables are considered for each segment. Moreover, there are N-1 wall segments and wall temperature variables.

For countercurrent and cocurrent heat exchangers, there are different boundary conditions. In cocurrent heat exchanger, the first node of the channel B is at fixed temperature, on the other hand, in countercurrent heat exchanger the last node of the channel B is at fixed temperature. Therefore, according to these arrangements initial conditions were also arranged for the nodes in between.

In order to analyze heat exchanger responses to temperature and mass flow rate, they were changed after a specific time when they reached steady state. The temperature of the cold fluid which is fluid A was increased by 1K after 8 seconds and its mass flow rate was increased by 0.1 kg/s after 15 seconds.

Boundary conditions for cocurrent heat exchanger:

Table 13: Boundary conditions for countercurrent heat exchanger

Temperature of fluid A at first node (N=1)	if time<8 then 300K else 301K
Temperature of fluid B at first node (N=1)	310K
Initial Temperature of fluid A and B at nodes 2…N	300K
Initial Temperature of wall segments	300K

Boundary conditions for countercurrent heat exchanger:

Table 14: Boundary conditions for cocurrent heat exchanger

Temperature of fluid A at first node (N=1)	if time<8 then 300K else 301K
Temperature of fluid B at node N	310K
Initial Temperature of fluid A and B at nodes 1…N-1	300K
Initial Temperature of wall segments	300K

Table 15: Simulation time of heat exchangers

Simulation time for both heat exchangers (s)
20

3.2.1. Plots of Heat Exchanger Models

At the steady state, as it is mentioned in the Chapter 2, total heat flow rates of Q_A and Q_B will be equal to steady state rate equation. Therefore, firstly total heat flow rates of Q_A and Q_B were verified with the steady state rate equation. It was observed that results were exactly matching in high N values while for low N values there were minor differences.In Table 16, steady state heat flow rates of cocurrent and countercurrent heat exchangers are given.

Table 16: Steady state heat flow rates

	Steady State Heat Flow Rate (W)	
Time (s)	**Countercurrent Mode**	**Cocurrent Mode**
time < 8	17004	15609.5
8 < time < 15	15303.6	14051
15 < time	15587.5	14396.2

In Figure 26, temperatures at the nodes of channel A and channel B are provided for the cocurrent heat exchanger in the cases for N=10 and N=1280. TA[1] and TB[1] are the fixed temperatures at the first node (inlet) of channel A and channel B respectively. TA[1] was changed by 1K after 8 seconds. And, after 15 seconds mass flow rate of fluid A was increased by 0.1 kg/s. TA[10] and TB[10] are the temperature variables at the last node (outlet) of channel A and channel B respectively for N=10 case. And, TA[1280] and TB[1280] are the temperature variables at the last node (outlet) of channel A and channel B respectively for N=1280 case.

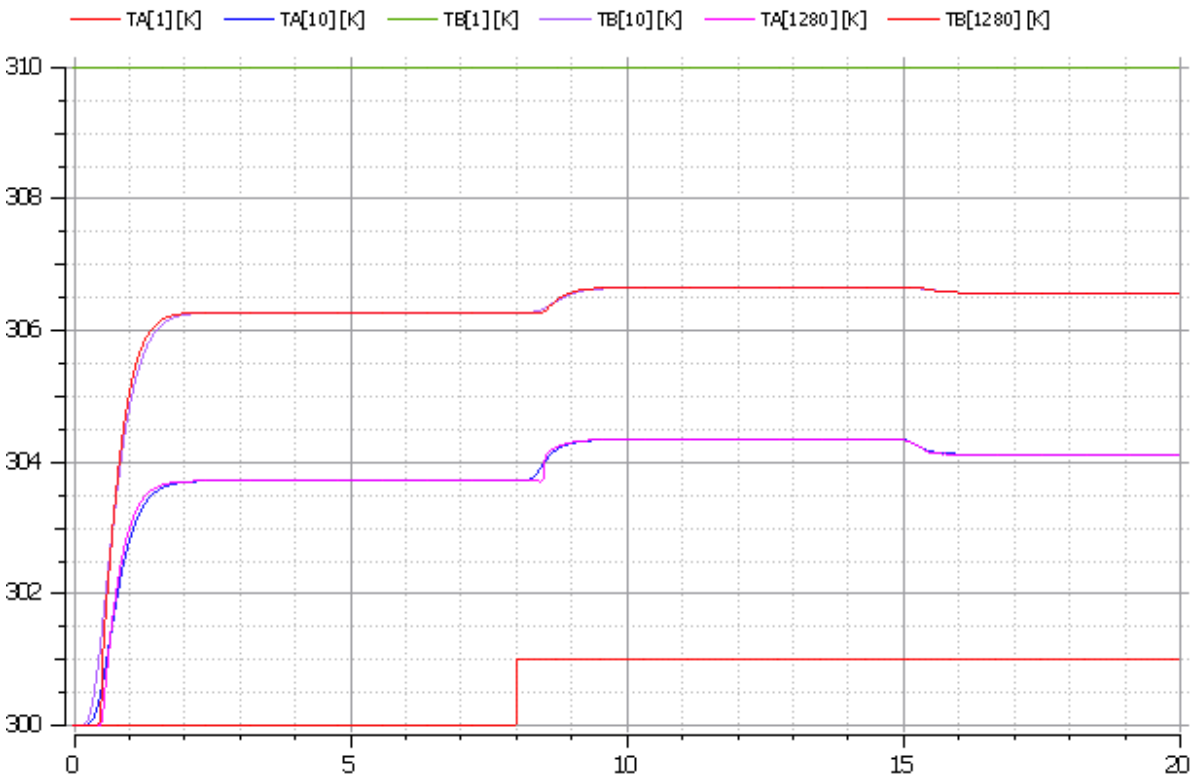

Figure 26: Cocurrent heat exchanger when N=10 and N=1280

In Figure 27, temperatures at the nodes of channel A and channel B are provided for the countercurrent heat exchanger for the cases N=10 and N=1280. In the figure, temperature variables that are shown are the last nodes (outlet) of channel A and the first nodes (outlet) of channel B. Moreover, boundary condition for the fluid A temperature is also shown being 300K at the beginning and after 8 seconds it increases to 301K. And, at the last node (inlet) of channel B, fluid B temperature was kept at 310K. TA[10] and TA[1280] are shown for channel A for the cases N=10 and N=1280 cases respectively. And, TB[1] variables are shown for channel B for the cases N=10 and N=1280. TA[10] and TA[1280] correspond to temperature variables at the last node (outlet) of channel A for N=10 and N=1280 respectively. In the figure, green TB[1] corresponds to the first node (outlet) of channel B when N=10. And, red TB[1] corresponds to the first node (outlet) of channel B when N=1280.

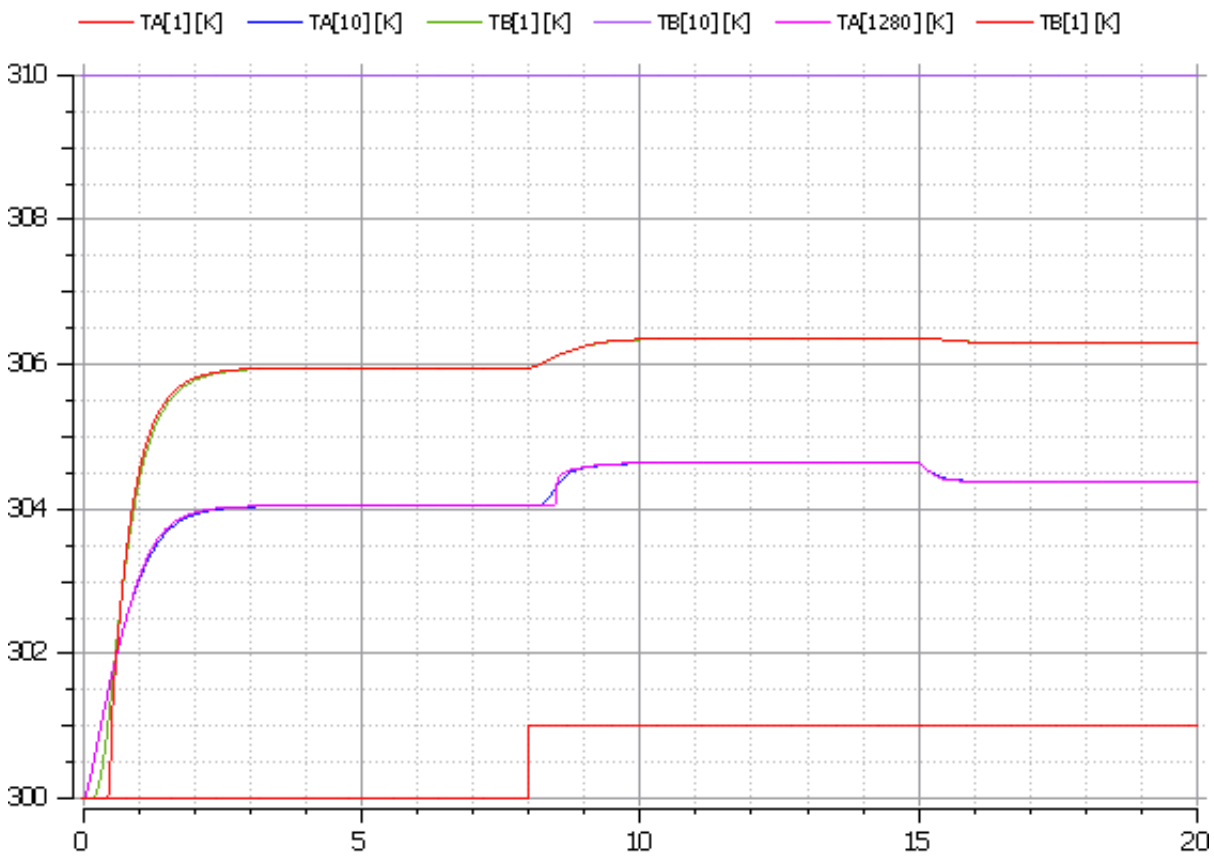

Figure 27: Countercurrent heat exchanger when N=10 and N=1280

As it is seen in Figure 26 and 27, temperature errors between the plotted nodes in the cases N=10 and N=1280 for both heat exchangers are very small. Because of this reason, only the temperature plots of N=10 and N=1280 are shown.

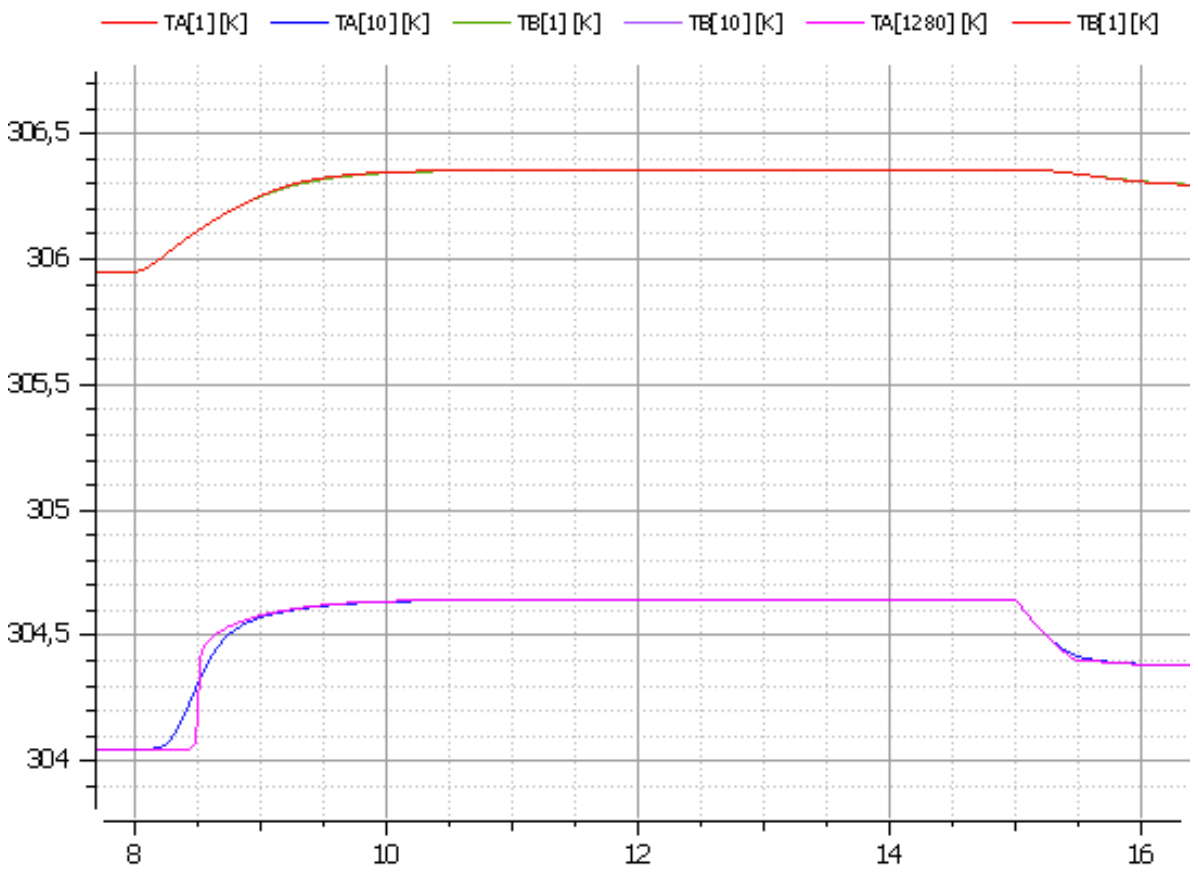

Figure 28: Zoomed version of Figure 27

In Figure 28, zoomed version of Figure 27 can be seen which shows the temperature variables of our interest. When the inlet temperature of channel A is increased by 1K after 8 seconds, its outlet temperature gets affected after a time delay because of the advective transport delay. However, mass flow rate increase of fluid A after 15 seconds affects the outlet temperature of channel A directly without a delay since it affects the entire channel at once. Furthermore, the same response is also observed for cocurrent heat exchanger in Figure 26. The difference between the cocurrent and countercurrent mode occurs due to the temperature variable values.

It was observed that countercurrent heat exchanger was more efficient than cocurrent heat exchanger. In countercurrent mode, temperature variables of hot fluid B decreased more whereas temperature variables of cold fluid A increased more. Therefore, total heat flow rate of the countercurrent heat exchanger was higher than the cocurrent heat exchanger.

3.2.2. Statistics of Heat Exchanger Models

Compilation and simulation times for countercurrent heat exchanger is given in Table 17.

Table 17: Statistics for countercurrent heat exchanger implemented by equations

N=10	**Compilation Time**	13,2865 s	**timeSimCode**	0.2041 s
			timeCompile	10.016 s
	Simulation Time	0,0452 s		
N=20	**Compilation Time**	13,2865 s	**timeSimCode**	0.4209 s
			timeCompile	10.8012 s
	Simulation Time	0,08825 s		
N=40	**Compilation Time**	16,9808 s	**timeSimCode**	0,9170 s
			timeCompile	11,5944 s
	Simulation Time	0,2059 s		
N=80	**Compilation Time**	24,0665 s	**timeSimCode**	2,0107 s
			timeCompile	14,1486 s
	Simulation Time	0,5891 s		
N=160	**Compilation Time**	42,7319 s	**timeSimCode**	4,5364 s
			timeCompile	18,9587 s
	Simulation Time	2,7388 s		
N=320	**Compilation Time**	116,235 s	**timeSimCode**	12,9859 s
			timeCompile	29,97 s
	Simulation Time	15,8217 s		
N=640	**Compilation Time**	460,1403 s	**timeSimCode**	34,629 s
			timeCompile	50,3959 s
	Simulation Time	118,512 s		
N=1280	**Compilation Time**	3189,9404 s	**timeSimCode**	125,6588 s
			timeCompile	91,3888 s
	Simulation Time	796,541 s		

Compilation and simulation times for cocurrent heat exchanger is given in table 18.

Table 18: Statistics for cocurrent heat exchanger implemented by equations

N=10	**Compilation Time**	13,0373 s	**timeSimCode**	0,2038 s
			timeCompile	10,5185 s
	Simulation Time	0,04765 s		
N=20	**Compilation Time**	14,2773 s	**timeSimCode**	0,429 s
			timeCompile	10,9431 s
	Simulation Time	0,0888 s		
N=40	**Compilation Time**	17,3529 s	**timeSimCode**	0,9172 s
			timeCompile	12,572 s
	Simulation Time	0,211 s		
N=80	**Compilation Time**	23,1271 s	**timeSimCode**	1,9963 s
			timeCompile	14,6171 s
	Simulation Time	0,6234 s		
N=160	**Compilation Time**	40,0048 s	**timeSimCode**	5,1013 s
			timeCompile	19,4841 s
	Simulation Time	2,9268 s		
N=320	**Compilation Time**	105,3304 s	**timeSimCode**	12,1592 s
			timeCompile	29,3973 s
	Simulation Time	18,7728 s		
N=640	**Compilation Time**	481,6894 s	**timeSimCode**	34,6752 s
			timeCompile	50,7986 s
	Simulation Time	141,603 s		
N=1280	**Compilation Time**	3304,775 s	**timeSimCode**	123,5185 s
			timeCompile	92,9118 s
	Simulation Time	954,558 s		

In Figure 29, compilation and simulation times are shown for cocurrent and countercurrent heat exchangers for our N values.

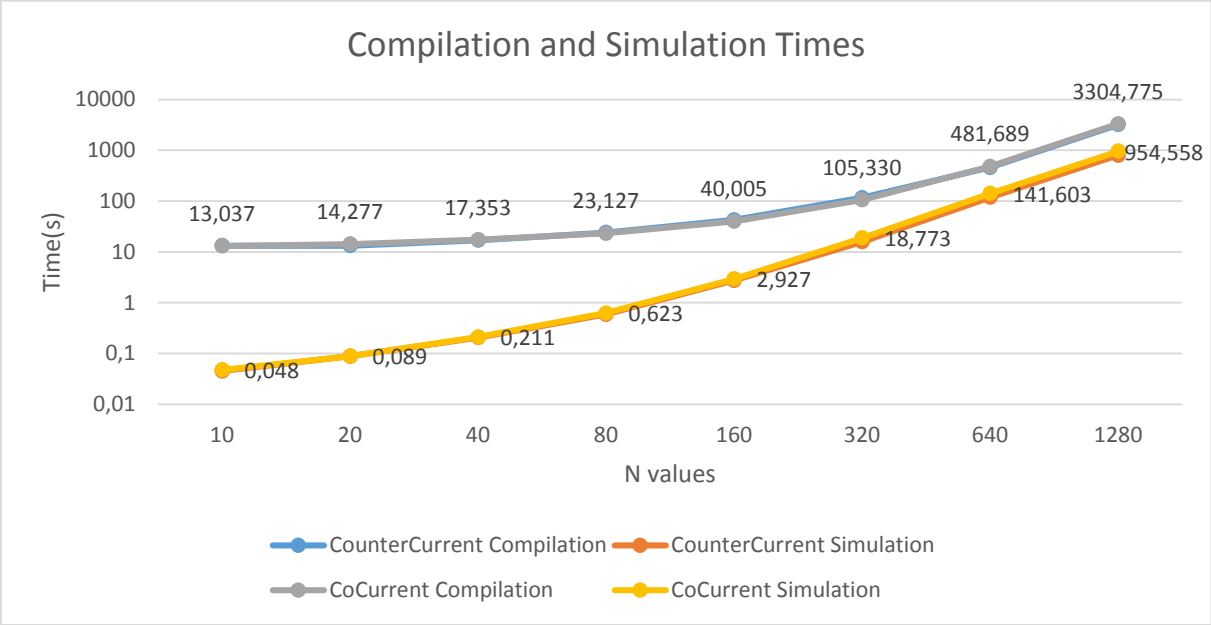

Figure 29: Compilation and simulation times of heat exchangers

Cocurrent and countercurrent heat exchangers at each N take the same time for compilation and for simulation. Compilation times of the models starts to increase after N=20. When N=640, the compilation time increases more than fourfold and when N=1280, compilation time increases almost sevenfold with respect to previous N compilation times. And, for large values of N, simulation times increase slightly with the third power of N.

3.3. Simulations of Flexible Beam Model

The simulations of the flexible beam were performed for 6 N values. Simulations were done for smaller N values comparing to other models because of its workload for OMC. The most important point in the plot analysis was to investigate the vibration frequency of the beam as N increases. The vibration frequency had to get closer to the analytical frequency as N increased.

An aluminum material was used for the flexible beam. Parameters are given in Table 19.

Table 19: Parameters for flexible beam

Length (m)	Height (m)	Width (m)	Density (kg/m^3)	Modulus of Elasticity (N/m^2)	Force (N)
0.5	0.02	0.05	2700	$6.9*10^{10}$	100

Using the parameters given in Table 19, parameters for the spring were obtained as shown in Table 20.

Table 20: Parameters for spring

Spring coefficient (N.m/rad)	Area moment of inertia (m^4)
$6.9*10^{10}*3.33*10^{-8}/(0.5/N)$	$3.33*10^{-8}$

Since the spring coefficient was a function of modulus of elasticity, area moment of inertia and a length of single element, it changed for each N value.

N values, which are the number of elements, used in the simulations are given in Table 21.

Table 21: N values for flexible beam simulation

N(Number of elements)
2
4
8
16
32
64

Table 22: Simulation time of flexible beam

Simulation Time (s)
0.15

3.3.1. Plots of Flexible Beam Model

Simulation time is 0.15 seconds for flexible beam models. As it is mentioned in the model explanation in Chapter 2, to the free end of the flexible beam a downwards force by 100 N was applied in the vertical axis between 0.001 and 0.002 seconds. And, the vibration of the flexible beam was observed. However, analytical solution represents the vibration of the free end directly starting from time 0 when a force of 100N is removed.

In Figure 30, red line represents the vibration of the free end in the case of analytical solution at the first mode, and blue line represents the vibration of the free end of the model implemented by MSL. In the figure, only the vibrations for N=64 are shown since for other N values vibrations showed the same behavior. The difference between the vibrations for different N values, occurred because of the vibration frequencies of the free end since N increases vibration frequency gets closer to the expected frequency.

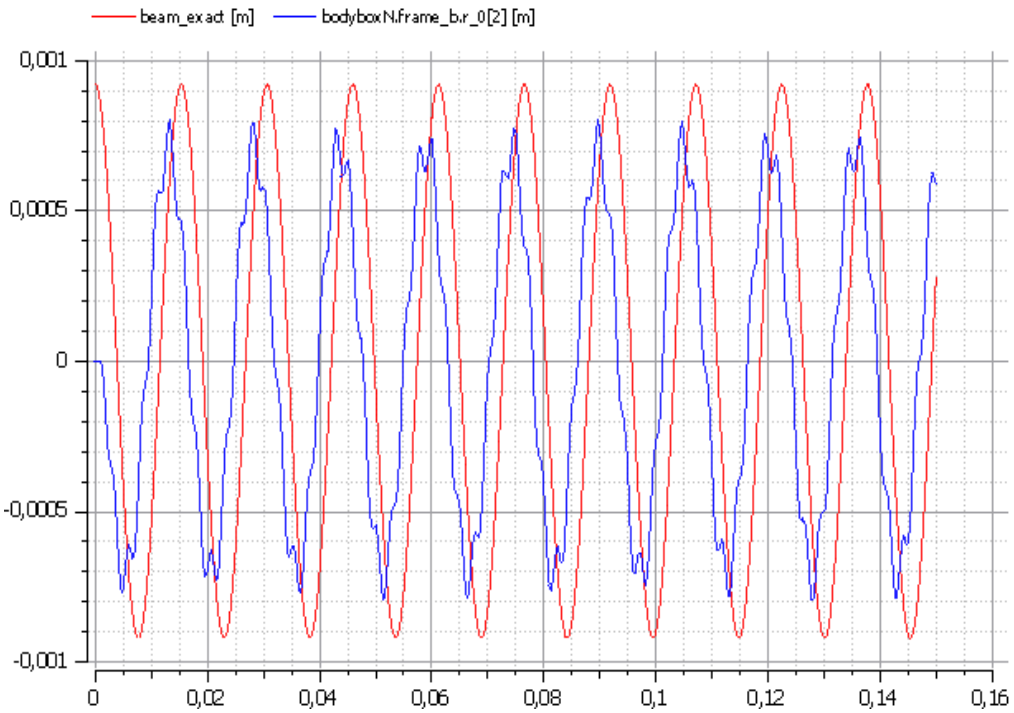

Figure 30: Vibration of the free end of the beam when N=64

In the figure, there is a phase shift between the vibrations of the flexible beam model and the analytical solution, this is because the flexible beam model was exposed to a force between 0.001 and 0.002 seconds and analytical solution starts directly vibrating from time 0.

As mentioned, the frequency of the vibration of the flexible beam at the free end was expected to get closer to the analytical calculated frequency as N increased. Approximate vibration frequencies of the free end of the flexible beam are given in the table below for each N according to the plots.

Table 23: Frequency values for corresponding N

N	Frequency of the first modes (Hz)
2	69,93
4	68,02
8	67,56
16	67,11
32	66,66
64	66,22

In the analytical solution, the frequency of the first mode was found to be 65.32 Hz. As it is seen in Table 23, as N increases frequency of the first mode of the flexible beam model is getting closer to the expected frequency. Therefore, for N goes to infinity, frequencies are expected to be equal.

3.3.2. Statistics of Flexible Beam Model

Compilation and simulation times are given below for the flexible beam model.

Table 24: Statistics for flexible beam implemented by MSL

N=2	**Compilation Time**	128,297 s	**timeSimCode**	1,6614 s
			timeCompile	11,7185 s
	Simulation Time	0,2651 s		
N=4	**Compilation Time**	135,2002 s	**timeSimCode**	2,0952 s
			timeCompile	12,9071 s
	Simulation Time	1,356 s		
N=8	**Compilation Time**	152,2238 s	**timeSimCode**	4,9046 s
			timeCompile	14,6112 s
	Simulation Time	5,1238 s		
N=16	**Compilation Time**	230,6504 s	**timeSimCode**	14,2493 s
			timeCompile	20,2011 s
	Simulation Time	46,59 s		
N=32	**Compilation Time**	578,2093 s	**timeSimCode**	46,81 s
			timeCompile	29,3969 s
	Simulation Time	298,312 s		
N=64	**Compilation Time**	3445,0252 s	**timeSimCode**	166,8684 s
			timeCompile	61,6683 s
	Simulation Time	2839,18 s		

In Figure 31, compilation and simulation times are shown for flexible beam for our N values.

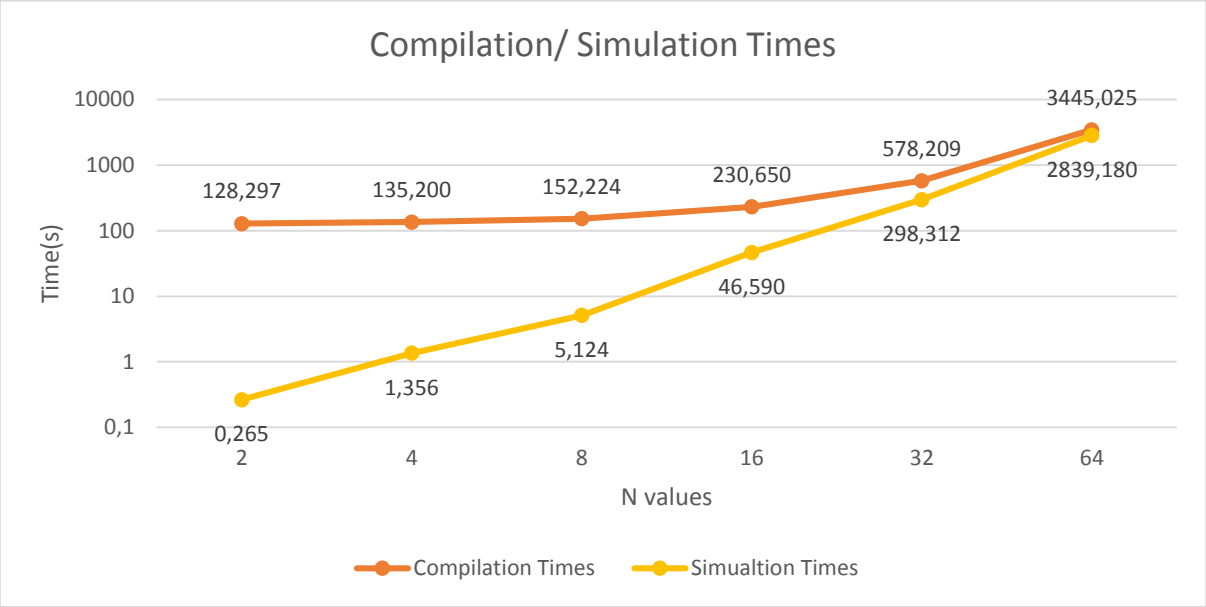

Figure 31: Compilation and simulation times of the flexible beam

Comparing the time spent of the other models for compilation and simulation, it was observed that the time spent for the flexible beam created by MSL was the highest. It can be observed that the multi body systems take a large compilation and simulation time due to its equations. Compilation time of the flexible beam model increase with slightly more than third power of N. And, simulation time also grows significantly as N doubles. An important point to mention is that when N=1280, total time spent for compilation and simulation is almost 2 hours.

3.4. Simulations of String model

As it is mentioned in the Chapter 2 in the explanation of the model of the string, N corresponds to number of revolute joints and there are N+1 body boxes. Therefore, length of the each body box is a function of N, corresponding to Length/ (N+1).

Parameters and simulation time for the string model:

Table 25: Parameters for string

Length(m)	Width(m)	Height(m)	Density(kg/m^3)	Damping coefficient(N.m.s/rad)
0.5	0.001	0.001	2000	10^{-5}

Table 26: Simulation time of string

Simulation Time(s)
0.8

3.4.1. Plots of String Model

In Figure 32 and 33, frame_b of some body boxes of the string are shown, frame_b corresponds to right end of a body box component. In the figures, a travelling wave is observed along the string model as expected. Two plots are provided for the cases when N=4 and N=64.

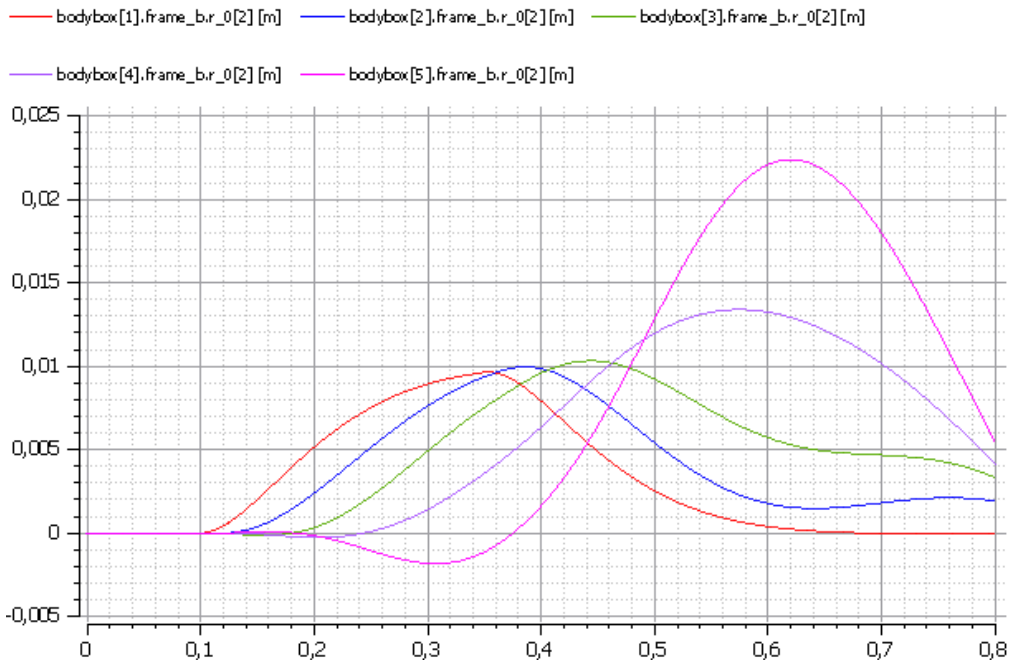

Figure 32: Plots of bodybox frames when N=4

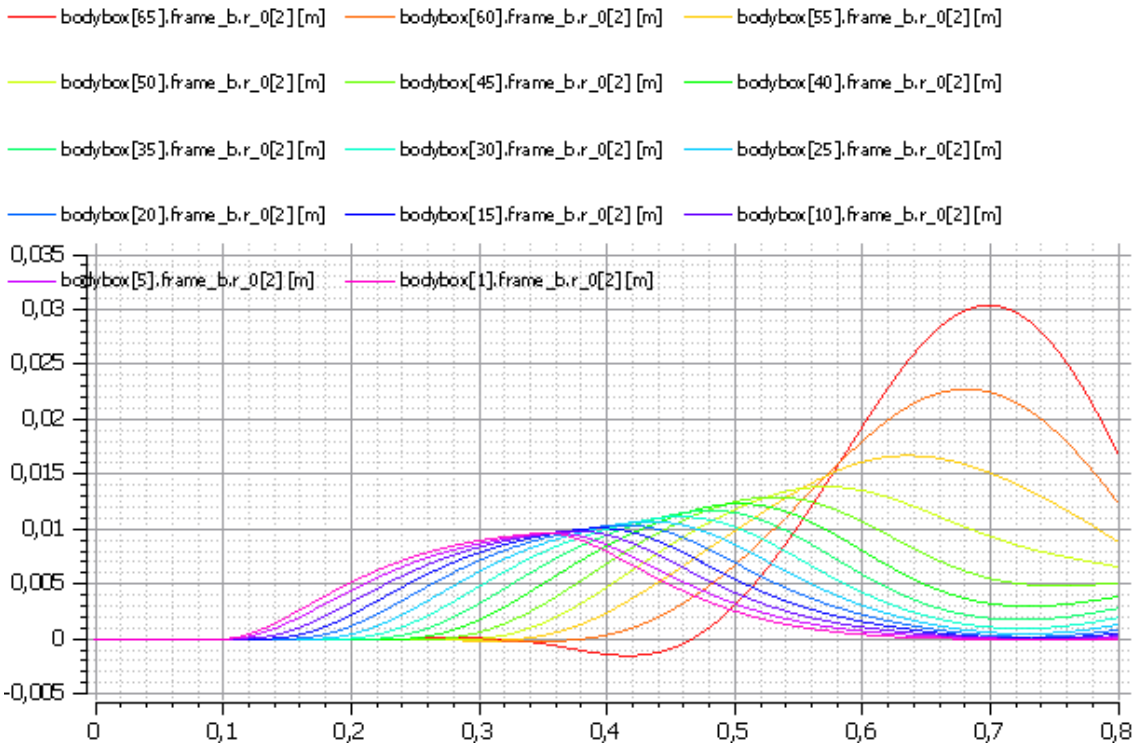

Figure 33: Plots of bodybox frames when N=64

3.4.2. Statistics of String Model

Compilation and simulation times of the string model are given in Table 27.

Table 27: Statistics for string

N=2	**Compilation Time**	155,9539 s	**timeSimCode**	1,8186 s
			timeCompile	14,4382 s
	Simulation Time	0,1341 s		
N=4	**Compilation Time**	159,0971 s	**timeSimCode**	3,5646 s
			timeCompile	14,9751 s
	Simulation Time	0,2242 s		
N=8	**Compilation Time**	171,5468 s	**timeSimCode**	7,453 s
			timeCompile	16,7655 s
	Simulation Time	0,3996 s		
N=16	**Compilation Time**	212,4966 s	**timeSimCode**	18,0573 s
			timeCompile	20,9911 s
	Simulation Time	1,0406 s		
N=32	**Compilation Time**	322,4101 s	**timeSimCode**	53,161 s
			timeCompile	30,7398 s
	Simulation Time	4,9838 s		
N=64	**Compilation Time**	684,1533 s	**timeSimCode**	182,7935 s
			timeCompile	62,2027 s
	Simulation Time	32,4428 s		

In Figure 34, compilation and simulation times of the string model are shown.

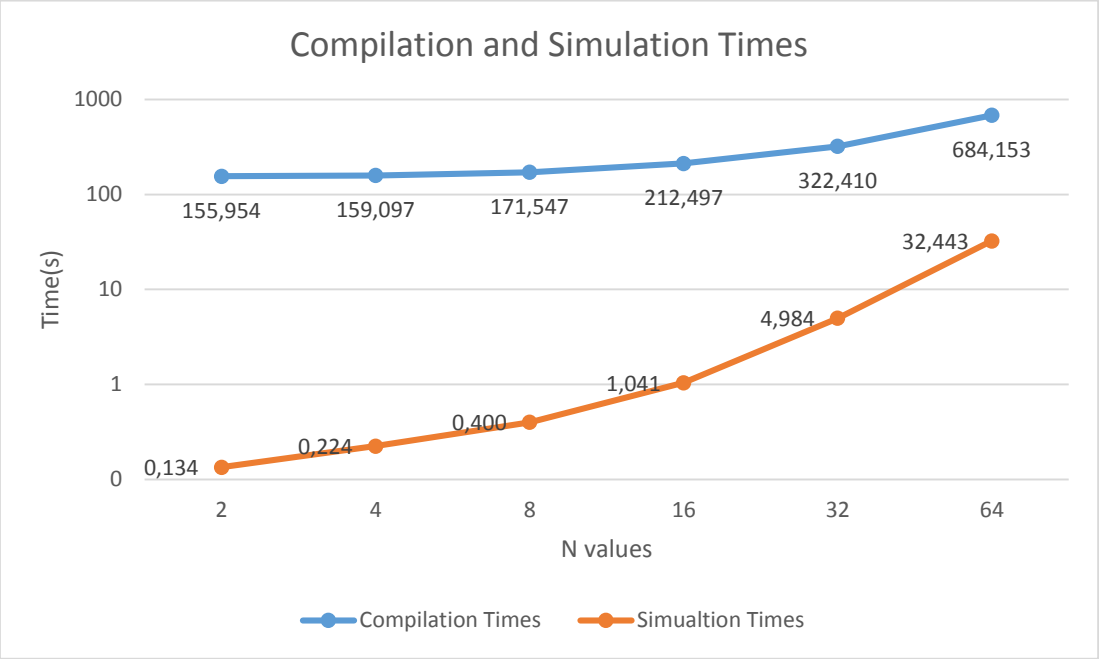

Figure 34: Compilation and simulation times for string

Compilation times of the string model took less time comparing to flexible beam when N=64 and N=1280 even though the two models were closely implemented. The difference between the models are the spring components and the uniform gravity. In the flexible beam, revolute joints were coupled with springs and dampers whereas in the string model, revolute joints were coupled with the dampers. In the string model, there was also uniform gravity.

3.5. Simulations of Transmission Line models

16 simulations were done for the transmission line circuit models for different N values, 8 for the transmission line circuit implemented by equations and 8 for the transmission line circuit implemented by Modelica.Electrical library. In transmission line models, N values correspond to number of segments. Parameters, N values and simulation time for the transmission line circuit models are given in the tables below.

Table 28: Transmission line parameters

Length(m)	r (Ω/m)	l (nH/m)	c(pF/m)	Z_o(Ω)	R_{load}(Ω)
100	48e-6	253	101	50	50

Table 29: Second order filter parameters

Gain	Damping	Cut-off frequency (rad/s)
1	1	$5*10^6$

Table 30: Values of N used for the transmission line simulations

N (number of segments in a transmission line)
10
20
40
80
160
320
640
1280

Table 31: Simulation time for transmission line models

Simulation Time(s)
4.10^{-6}

Using the parameters, time delay of the transmission line was found $5{,}055.10^{-7}$ seconds.

3.5.1. Plots of Transmission Line Models

For the verification of the models, Figure 35 is given in order to show that transmission line circuit by Electrical library and equations are matching. For the verification, N=1280 is used since it was the most accurate. Moreover, green line (vol[1]) represents the filter output voltage which is the input voltage of the transmission line. Blue (vol[1280]) and red(resistor.p.v) lines represent the voltage on the load resistance in the case of transmission line circuit implemented by equations and Electrical library respectively. It can be seen that blue and red lines are exactly matching and time delay between the input voltage and the output voltage of the transmission line is matching with the theoretical time delay:$5{,}055.10^{-7}$ seconds.

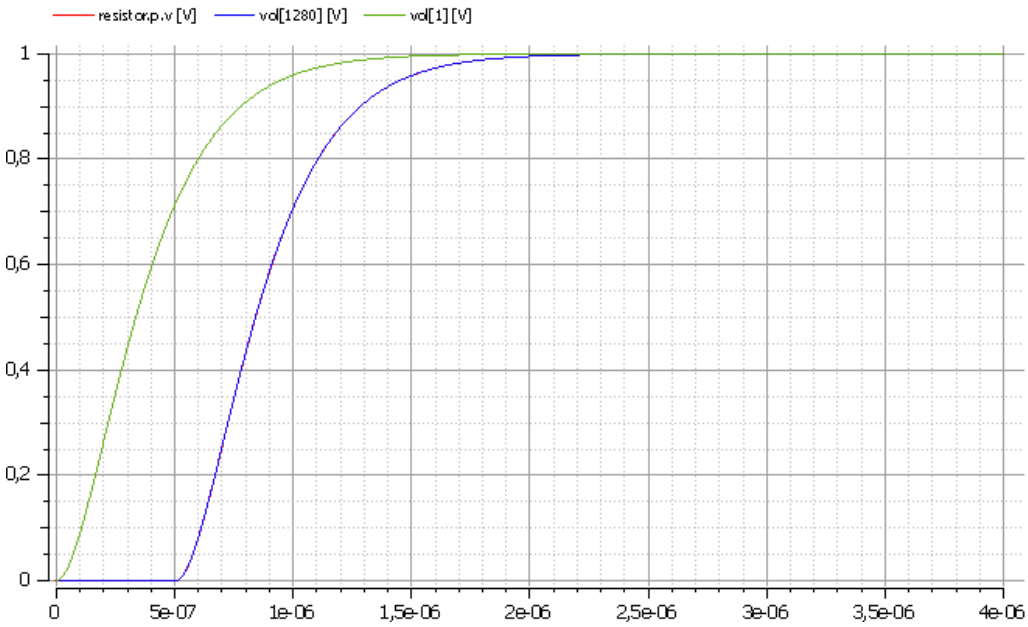

Figure 35: Transmission Line in the case of N=1280

In Figure 36 and 37, output voltages of the transmission line circuits implemented by equations and MSL are given for the different N values which are shown in Table 30. In both figures, all the lines represent the output voltage of the transmission line, thus, the voltage on the load resistor.

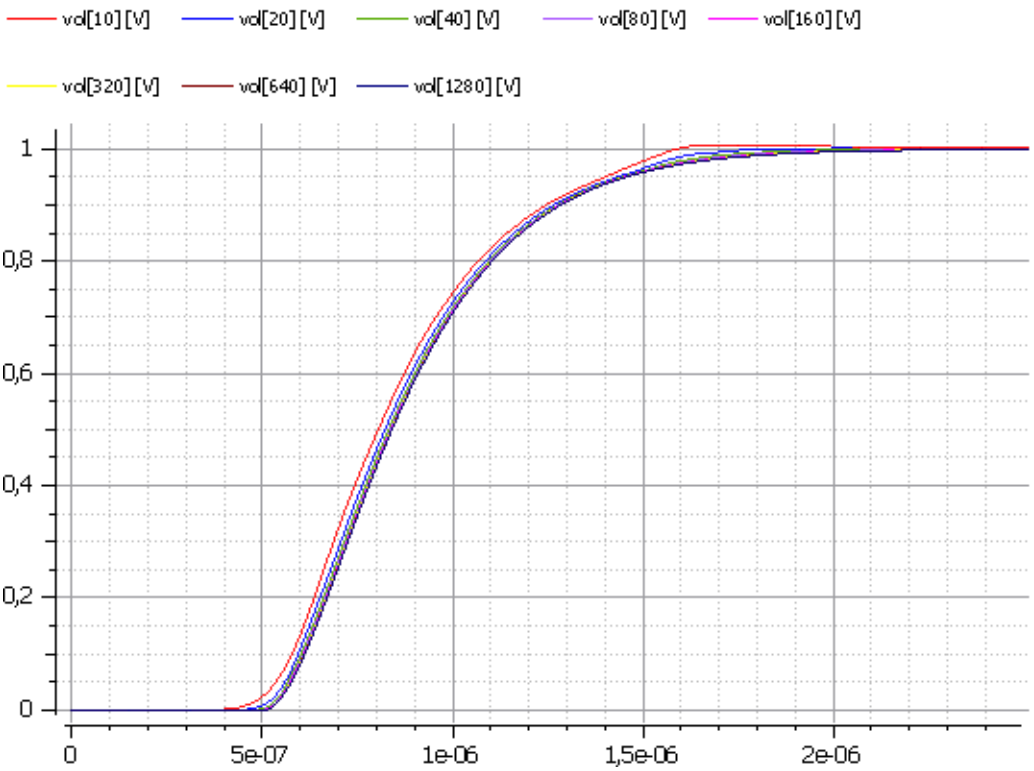

Figure 36: Output voltages of transmission line implemented by equations

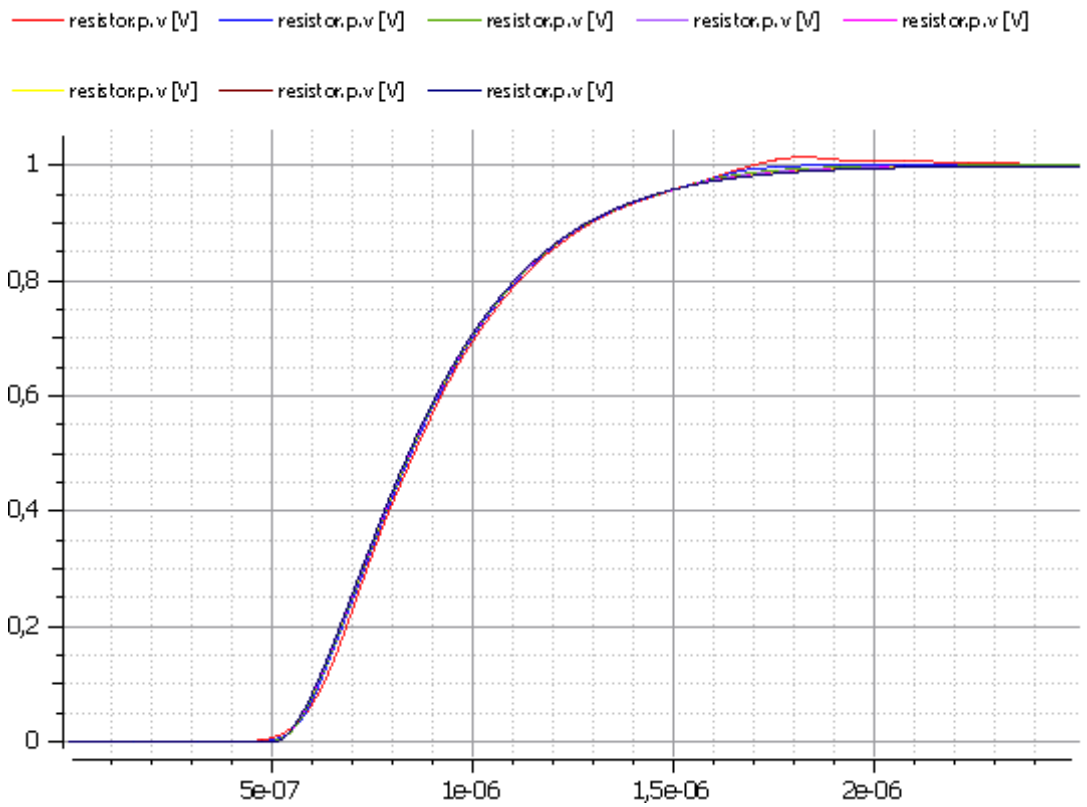

Figure 37: Output voltages of transmission line implemented by MSL

In the figures, as N increases output voltage error decreases. It was observed that starting from N=160 two models were meeting the expected output voltage response. Furthermore, considering the time delay as calculated $5{,}055.10^{-7}$ seconds, transmission line circuit created by MSL is more accurate than the model implemented by equations. Starting from N=20, it satisfies the theoretical time delay while it does not show the exact behavior. Red lines in both figures represent the output voltage of the transmission line when N=10, both figures have some oscillations when they about to reach steady state, however, the model implemented by MSL is closer to the expected result in terms of time delay and the models implemented by MSL have less error.

3.5.2. Statistics of Transmission Line Models

Compilation times and simulation times are provided for each model of the transmission line circuit for 8 different values of N.

Statistics for the transmission line circuit implemented by equations is given in Table 32.

Table 32: Statistics for transmission line circuit implemented by equations

N=10	**Compilation Time**	18,227 s	**timeSimCode**	0,094911 s
			timeCompile	14,9981 s
	Simulation Time	0,0918539 s		
N=20	**Compilation Time**	18,1851 s	**timeSimCode**	0,16728 s
			timeCompile	14,5725 s
	Simulation Time	0,119774 s		
N=40	**Compilation Time**	19,1862 s	**timeSimCode**	0,3383 s
			timeCompile	15,06125 s
	Simulation Time	0,2695 s		
N=80	**Compilation Time**	22,1957 s	**timeSimCode**	0,70748 s
			timeCompile	16,0989 s
	Simulation Time	1,11 s		
N=160	**Compilation Time**	33,4538 s	**timeSimCode**	1,58155 s
			timeCompile	19,8994 s
	Simulation Time	4,57905 s		
N=320	**Compilation Time**	71,9591 s	**timeSimCode**	3,89609 s
			timeCompile	25,1333 s
	Simulation Time	32,1252 s		
N=640	**Compilation Time**	172,3209 s	**timeSimCode**	8,8533 s
			timeCompile	29,1568 s
	Simulation Time	117,686 s		
N=1280	**Compilation Time**	1392,8 s	**timeSimCode**	27,5935 s
			timeCompile	48,3279 s
	Simulation Time	1273,45 s		

Statistics for the transmission line circuit implemented by MSL is given in Table 33.

Table 33: Statistics for transmission line circuit implemented by MSL

N=10	**Compilation Time**	20,5265 s	**timeSimCode**	0,2732 s
			timeCompile	16,4346 s
	Simulation Time	0,0703 s		
N=20	**Compilation Time**	18,3549 s	**timeSimCode**	1,07107 s
			timeCompile	12,5433 s
	Simulation Time	0,1132 s		
N=40	**Compilation Time**	22,4014 s	**timeSimCode**	0,9868 s
			timeCompile	13,5865 s
	Simulation Time	0,2648 s		
N=80	**Compilation Time**	32,1239 s	**timeSimCode**	2,4577 s
			timeCompile	14,8782 s
	Simulation Time	0,757 s		
N=160	**Compilation Time**	74,2713 s	**timeSimCode**	8,3692 s
			timeCompile	27,5583 s
	Simulation Time	4,2633 s		
N=320	**Compilation Time**	208,8679 s	**timeSimCode**	38,1643 s
			timeCompile	33,5111 s
	Simulation Time	22,8354 s		
N=640	**Compilation Time**	934,1011 s	**timeSimCode**	202,9418 s
			timeCompile	62,8952 s
	Simulation Time	141,316 s		
N=1280	**Compilation Time**	6071,9205 s	**timeSimCode**	1562,8448 s
			timeCompile	131,4872 s
	Simulation Time	1568,27 s		

In Figure 38, compilation times of transmission line circuit implemented by equations and MSL according to our N values is shown.

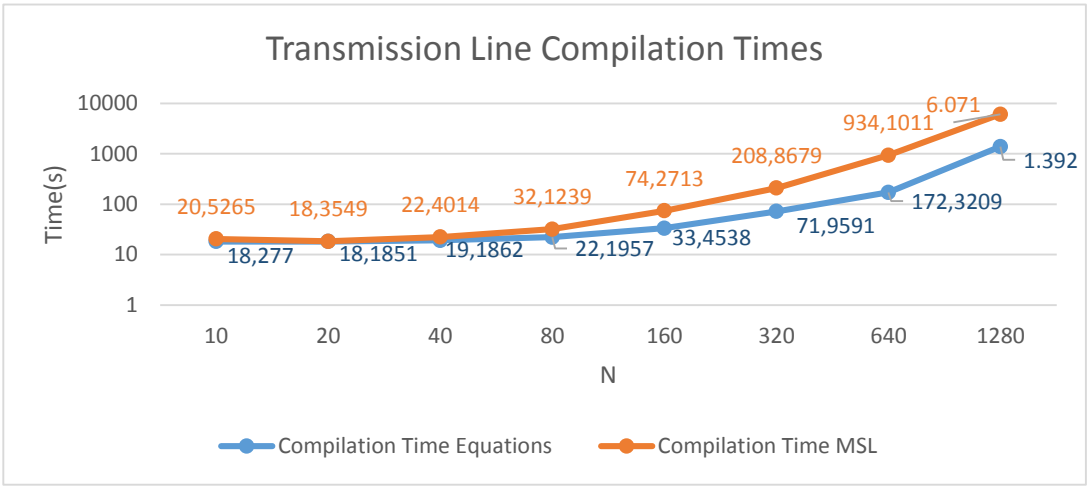

Figure 38: Compilation times of transmission line circuit models

The compilation times of the two models are close until N=40. However, as N increases after N=40, compilation times for both models are increasing and the time difference between the two models are growing. The model which is implemented by MSL takes more time to compile. And, its compilation times increase roughly with second power of N. Moreover, when N=1280, the time that takes to compile the model implemented by MSL is around 1.6 hours. However, when N=1280, the model implemented by equations takes around 23 minutes.

In Figure 39, simulation times of transmission line circuit implemented by equations and MSL are shown. For higher of values of N, simulation time of the transmission line circuit model implemented by MSL increases roughly with third power of N.

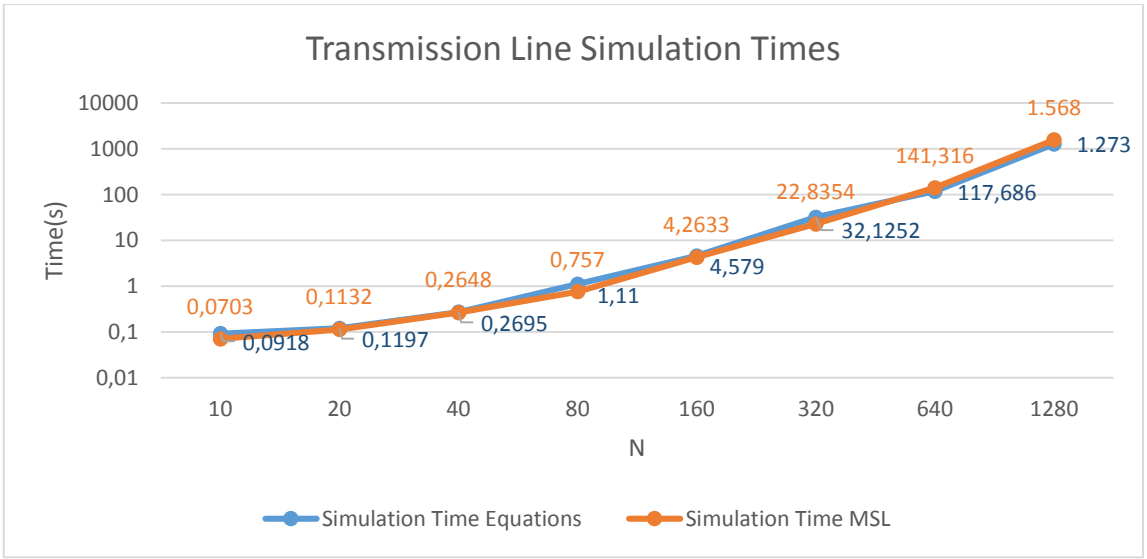

Figure 39: Simulation times of transmission line circuit models

4. Conclusion and Future Work

A test suite of large scalable models from different domains have been implemented in order to test performance of OpenModelica and other Modelica compilers. The scope of this test suite will be enlarged in future by other people who are interested in this research area.

All the models were tested for gradually increasing N values, hence, discretization. In simulations, as expected it was observed that as N increased in the models, their plots reflected more accurate results. However, this yielded more equations and increased compilation and simulation times. The models which were implemented using equations were faster than the models implemented using MSL in terms of compilation and simulation times for increasing N. However, in terms of accuracy, MSL showed better results than equations for lower N values.

It was observed that mechanical models which were implemented using MultiBody library caused more compilation and simulation times comparing to Electrical and Thermal libraries. The reason is the number of equations because multi body systems require the solution of the motion equations for each component. Therefore, it requires a large portion of time. Moreover, Thermal library was observed to be quicker comparing to Electrical library in compilation and simulation.

Compiler and sequential simulation codes became less efficient while tackling large models, especially in mechanical domain, implemented by MSL and it caused to spend a lot of time for the compilation and simulation of some models. Therefore, compiler and sequential simulation code need to be improved in order to support large models.

Multi rate and sparse solvers, and parallelization algorithms have been improved by developers to decrease the simulation times. Parallelization enables the modeler to split the equations into several independent parts. A module has already been implemented for OMC which enables to use different scheduling algorithms for parallelization. Therefore, the models can be tested with parallelization algorithms to observe whether it is possible to get faster results.

References

[1] Openmodelica.org. *Welcome to Open Modelica – OpenModelica*. [Online]. Available at: https://openmodelica.org.

[2] Fritzson, P. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley 2014.

[3] Cellier, F.E., Kofman, E. *Continuous System Simulation*. Springer. 2006.

[4] Polyanin, Andrei D. *Handbook of Linear Partial Differential Equations for Engineers and Scientists*. Chapman & Hall/CRC, 2002. Politecnico Di Milano. Web. 14 Jan. 2015.

[5] Subramanian, R. *Thermal Analysis of a Steady State Heat Exchanger.* Lecture notes at Clarkson University. [Online] Available at: http://web2.clarkson.edu/projects/subramanian/ch330/notes/Thermal%20Analysis%20of%20a%20Steady%20State%20Heat%20Exchanger.pdf.

[6] Open Source Modelica Consortium. *OpenModelica System Documentation Version 2014-02-01 for OpenModelica 1.9.1 Beta1*. February 2014.

[7] Skoglunda Tomas, Årzén Karl-Erik, Dejmek Petr. *Dynamic object-oriented heat-exchanger models for simulation of fluid property transitions*. Lecture notes at Lund University [Online]. Available at: http://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=406058&fileOId=581724.

[8] Chud Victor, Mukherjee Arnav, Wendlandt Jeff, Kennedy Dallas. Modeling Flexible Bodies in SimMechanics. *Technical Articles and Newsletters for MathWorks*. 2006. [Online] Available at: http://it.mathworks.com/company/newsletters/articles/modeling-flexible-bodies-in-simmechanics-and-simulink.html.

[9] Bruni, Stefano. *Lesson 7 Dynamics of continuous systems- bending vibrations of beams*. Lecture notes at Politecnico di Milano. 2015.

[10] Corradi, Roberto. *Axial and Bending Vibration of rectilinear beams.* Lecture notes at Politecnico di Milano.2014.

[11] Walther Marcus, Waurich Volker, Schubert Christian, Dr.-Ing. Gubcsh Ines. Equation based parallelization of Modelica models. *Proceedings of the 10th International Modelica Conference*. Lund, Sweden. 2014.

[12] Whitney, Scott. *Vibrations of Cantilever Beams: Deflection, Frequency, and Research Uses*. Web Pages on Mechanics at University of Nebraska-Lincoln. April 23, 1999. [Online] Available at: http://emweb.unl.edu/Mechanics-Pages/Scott-Whitney/325hweb/Beams.htm.

[13] Volterra E., Zachmanoglou, E. C. *Dynamics of Vibrations*. Columbus, Charles E. Merrill Books, Inc., 1965.

[14] Frenkel Jens, Schubert Christian, Kunze Gübert, Fritzson Peter, Sjölund Martin, Pop Adrian. Towards a Benchmark Suite for Modelica Compilers: Large Models. *8th International Modelica Conference 2011.* Germany. March 20-22, 2011. [Online] Available at: https://modelica.org/events/modelica2011/Proceedings/pages/papers/07_1_ID_183_a_fv.pdf.

[15] Aronsson Peter, Fritzson Peter. Multiprocessor Scheduling of Simulation Code From Modelica Models. *2nd International Modelica Conference*. Germany. March 18-19, 2002. [Online] Available at: https://modelica.org/events/Conference2002/papers/p41_Aronsson.pdf.

[16] Wikipedia, (2015). *Transmission Line.* [Online] Available at: http://en.wikipedia.org/wiki/Transmission_line