Exploiting OpenMP in the Initial Section of Modelica Models (Work in Progress)

Javier Bonilla[1] Luis J. Yebra[1] Sebastián Dormido[2]

[1]PSA-CIEMAT, Plataforma Solar de Almería - Centro de Investigaciones Energéticas, MedioAmbientales y Tecnológicas, Almería, Spain, {javier.bonilla,luis.yebra}@psa.es

[2]UNED, Universidad Nacional de Educación a Distancia, Madrid, Spain, sdormido@dia.uned.es

Abstract

This paper presents a practical case where parallelization for multi-core processors can be exploited in Modelica models using OpenMP. Although this parallelization is applied to a particular case and to a particular section in the code, the parallel implementation is straightforward and a gain in speed of around 11% is obtained. The particular section in the code is the initial section and the particular case is to consider that the initial section is time consuming and the operations are independent from each other. The particular case is tested in a real dynamic model during the simulation and calibration processes. The most appealing feature of this method is that it is straightforward to implement and that it could be easily adopted by equation-based modelling languages. On the other hand, the process is not automatically performed and the modeller needs to have a minimum knowledge about parallel computing.

Keywords Modelica, OpenMP, parallelization, initial section, multi-core processors

1. Introduction

Nowadays, modern equation-based object-oriented (EOO) modelling languages are continuously increasing their expressiveness to describe and model complex systems. However, there is an important drawback of having large and complex system models, the required computational effort to simulate is very high.

Commonly, EOO models are compiled as single-threaded executables not taking advantage of the newest multi-core processors available even on desktop computers. Moreover, and considering the particular case of Modelica [13], the extension of the language to consider multi-core processors is not easy due to the flattening of the equation-based object-oriented Modelica code into C code.

4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. September, 2011, ETH Zürich, Switzerland.

With regard to parallelization in EOO modelling languages it is worth mentioning [3, 4, 12, 19]. This paper does not pretend to be a meticulous study about parallelization in EOO modelling languages, it just describes a straightforward parallelization method in the initial section of the resulting C source code obtained from the Modelica model, and not considering the parallelization in the numerical integration process.

2. OpenMP

OpenMP (Open Multi-Processing) [2] is an application programming interface (API) for multi-platform (Unix and Microsoft Windows platforms) shared-memory parallel programming in C/C++ and Fortran. It provides a simple and flexible interface to develop parallel applications by means of a set of compiler directives, library routines, and environment variables. Its applicability ranges from desktop computers to clusters and supercomputers.

OpenMP is maintained by the OpenMP Architecture Review Board (ARB). The first OpenMP API specifications was released in 1997 for Fortran 1.0 whereas for the C/C++ languages was released the following year, in 1998. The OpenMP version 2.0 was released in 2000 and 2002 for Fortran and C/C++ respectively. In 2005, both versions were unified in version 2.5. Version 3.0 was released in 2008. The current version, version 3.1, has been recently released in June, 2011.

The key concept in OpenMP is multithreading, in this method of parallelization the master thread forks a series of slave threads for particular parallel tasks, the run-time environment is responsible of allocating the threads to different processors or cores. Figure 1 illustrates this concept, where the master thread forks 2 additional threads for tasks I, 3 additional threads for task II and 1 additional thread for task III.

The core elements of the OpenMP API are summarized and briefly explained in the following list.

- **Parallel control directives.** They control the flow execution of the program (i.e. *parallel* directive).
- **Work sharing.** They distribute the execution of instruction among the processors or cores (i.e. *parallel for* or *section* structures).

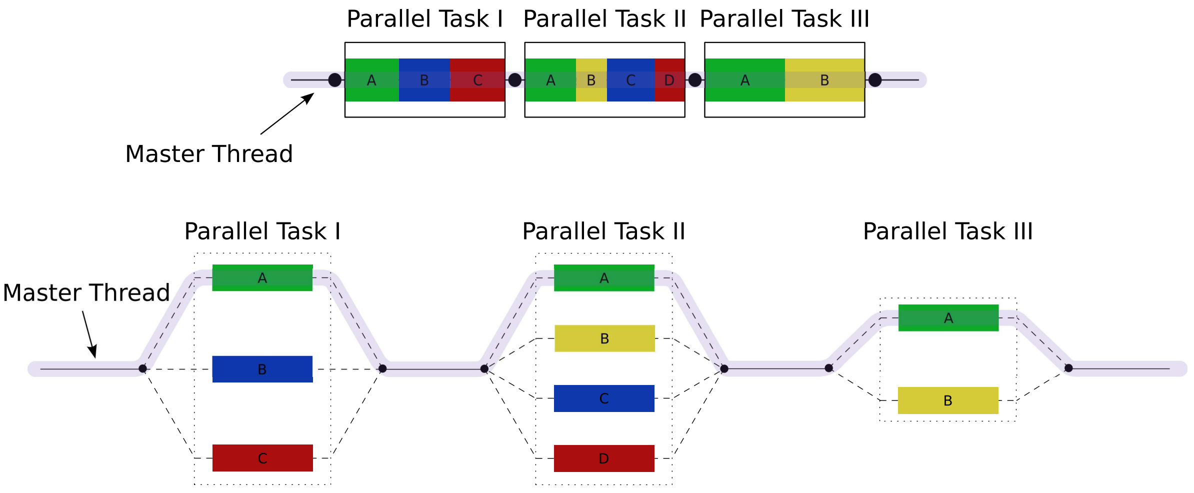

Figure 1. OpenMP multithreading concept [1].

- **Data environment.** It is defined by the scope variables, *shared* and *private*.
- **Synchronization.** It is controlled by the *critical*, *atomic* and *barrier* directives.
- **Run-time functions.** Set and provide run-time information (i.e. *omp_set_num_thread()* and *omp_get_num_thread()* which set and obtain the maximum number of threads that can be allocated).

3. Real System Under Study

This section explains briefly the facility under study, the developed Modelica model and the defined simulator scheme to facilitate the simulation, calibration and validation processes.

3.1 DISS facility: Direct Steam Generation Parabolic-Trough Solar Thermal Power Plant

The real system under study is the CIEMAT-PSA (Centro de Investigaciones Energéticas Medioambientales y Tecnológicas - Plataforma Solar de Almería, a Spanish government research and test center) DISS (DIrect Solar Steam) test facility, a parabolic-trough solar thermal power plant. A general view of this facility is shown in Figure 2.

The parabolic-trough technology is one of the several different solar thermal concentrating technologies available. Parabolic-Trough Collectors (PTCs) are solar concentrators which convert the direct solar radiation into thermal energy, heating a heat transfer fluid (HTF) up to around 675 K. Their high working temperature makes PTCs suitable for supplying heat to industrial processes, replacing traditional fossil fuels [11] [18].

The HTF used in the DISS test facility is the two-phase-flow steam-water, which circulates in three different states, subcooled liquid, steam-water mixture and superheated steam.The aim of the DISS facility is to develop a new generation of solar thermal power plants using parabolic-trough collectors to produce high pressure steam in the absorber tubes, thus eliminating the oil commonly used as a heat transfer medium between the solar field and the conventional power block. This kind of technology is known as Direct Steam Generation (DSG), it increases overall system efficiency while reducing investment costs.

3.2 Modelica model

A DISS solar thermal power plant equation-based object-oriented dynamic model was developed to study the behavior of the real plant [17]. An EOO methodology was chosen in order to describe the system as a set of equations which are acausal, maintaining it mathematical meaning. Moreover, a object-oriented methodology allows to define basic models which can be reused to develop new complex models without additional effort [5]. This methodology allows to develop reusable and easy to maintain components.

The modelling language chosen was Modelica. Modelica is developed and maintained by the Modelica Association. Modelica is a suitable modelling language to develop complex mathematical models of physical systems. Modelica also has useful libraries to develop thermo-hydraulic systems. Modelica Media [14] is a thermodynamic properties computation library, which follows IAPWS (the International Association for the Properties of Water and Steam) recommendations in its latest IF97 formulation, (Industrial Formulation 1997) [10]. This formulation is optimised for short computing times and low CPU load. Furthermore, the ThermoFluid library [16, 8] provides a framework and basic components for modelling thermo-hydraulic and pro-

Figure 2. General view of the DISS test facility owned by Plataforma Solar de Almería (CIEMAT).

cess systems in Modelica. The Integrated Development Environment (IDE) chosen, which supports the Modelica language, has been Dymola [7].

Figure 3 shows the DISS test facility Modelica component diagram which has 11 PTC components. The model inputs are: the ambient temperature ($Tamb$), the solar radiation (Rad), regarding the HTF, the inlet temperature ($inletTemp$), the inlet pressure ($inletPres$) and the mass flow ($mdot\_ws$) of the fluid, the three last inputs are also provided for the last PTC injector.

3.3 Simulator scheme

With the aim of facilitating the settlement of the initial conditions for the simulation, the calibration and validation processes, a simulator scheme was developed. Figure 4 shows the DISS facility simulator scheme which is not only the model itself but a series of tools to facilitate the simulation.

Among the developed tools in the simulator scheme, it is worth mentioning the following.

- A web application to easily access the real data from the data acquisition system (DAS) with the aim of comparing the real data with the simulation results and also to obtain the initial conditions for the dynamic simulation.
- A computer program to convert the real data obtained from the web application to a input trajectory file used in the Dymola IDE.
- A Modelica library to retrieve the initial values from the trajectory file and solve the initial condition problem. This Modelica library reads the input trajectory file to set values to certain parameters which are used to solve the initial condition problem. These reading operations are time consuming and they can be easily parallelized.

The calibration process has been performed using Matlab/Simulink [15]. Dymola includes mechanisms to export Modelica models to Simulink in a easy and direct way using a Simulink block where parameters and inputs can be defined. The Matlab Genetic Algorithm Toolbox [6] has been used to calibrate the DISS model, a multi-objective approach was selected in order to minimize the absolute value of the percentage relative error between the real and simulated output temperatures for each PTC. Figure 5 shows the DISS Simulink block in Matlab. More information about calibration of Modelica models in Matlab together with a practical example can be found in [9].

4. Initial Section Parallelization

As previously explained in section 3.3, a Modelica library was developed to retrieve real data from the input trajectory file with the aim of setting several parameters to solve the initial condition problem. Reading from a text file is a time consuming operation. Some data must be read from this input file, including the initial inlet and outlet pressure, temperature, mass flow of the HTF, etc. These reading operations can be easily parallelized because they are independent from each other.

The procedure to parallelize the initial section in the resulting C code, obtained from the translation of the Modelica model by the Dymola tool, is the following.

1. In the resulting C source code generated by the Dymola tool (*dsmodel.c*), it has been included a reference to the OpenMP header (*omp.h*), as shown in Listing 1, in order to use in the source code, the OpenMP API, directives and functions.

```
#include <omp.h>
```

Listing 1. OpenMP header inclusion

2. Set the maximum number of threads to be executed using the *omp_set_num_threads()* function as shown in Listing 2. In our particular case this value was set considering the number of the processor cores, 4 cores.

```
constant int NCores = 4;
omp_set_num_threads(NCores);
```

Listing 2. Setting the maximum number of threads

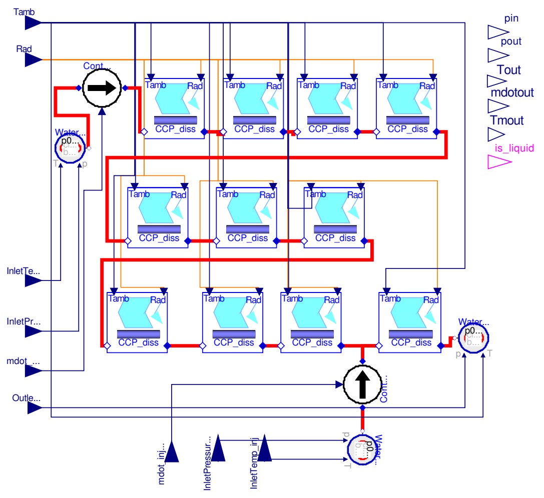

Figure 3. DISS test facility Modelica model.

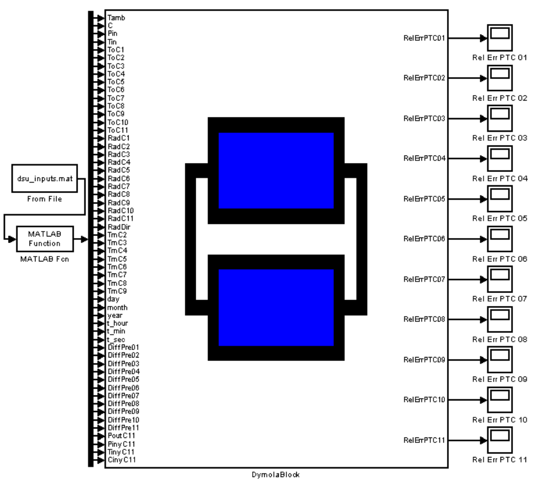

Figure 5. DISS Simulink block.

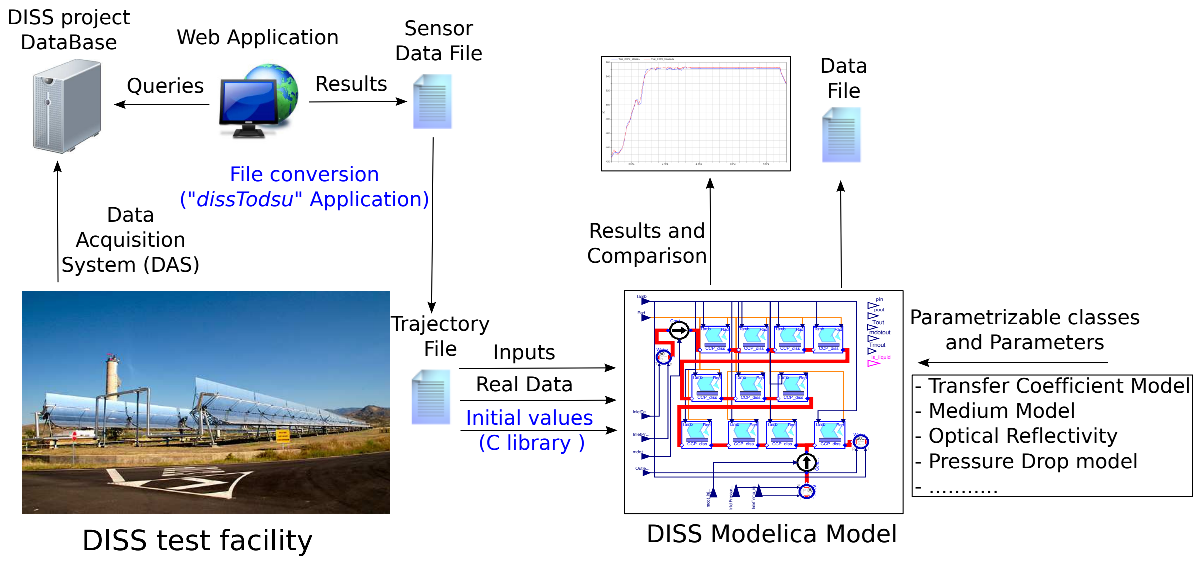

Figure 4. DISS simulator scheme.

```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    {
      ........
      ........
     T[0]=getIniValue(iniTime,"Tem PTC1");
     T[1]=getIniValue(iniTime,"Tem PTC2");
      ........
      ........
    }
    #pragma omp section
    {
      ........
      ........
      p[0]=getIniValue(iniTime,"Pres PTC1");
      p[1]=getIniValue(iniTime,"Pres PTC2");
      ........
      ........
    }
    #pragma omp section
    {
      ........
      ........
      r[0]=getIniValue(iniTime,"Rad PTC1");
      r[1]=getIniValue(iniTime,"Rad PTC2");
      ........
      ........
    }
    #pragma omp section
    {
      ........
      ........
     m[0]=getIniValue(iniTime,"Mflow PTC1");
     m[1]=getIniValue(iniTime,"Mflow PTC2");
      ........
      ........
    }
  }
}
```

Listing 3. OpenMP parallel sections

3. Locate the initial section/s in the *dsmodel.c* file which can be identified in the C source code by the *InitialSection* identifier.

   After that, the sentences in the initial section must be manually distributed between the different threads to balance the computational load between cores. For that purpose the OpenMP *sections* directives were used, the source code is shown in Listing 3.

   First, the *parallel* directive was used to indicate that the following sentences must be executed in parallel, then the *sections* and *section* directives were used to define 4 sections which correspond with 4 threads. The reading operations were equally distributed between the 4 sections, each one executed by a different thread in each available processor core.

4. Compile the *dsmodel.c* including the OpenMP library. For the GNU Compiler Collection (GCC) in Linux operating systems, only the *-fopenmp* modifier is necessary to compile the *dsmodel.c* source code with OpenMP support. For that purpose, a script file, *compile.sh*, was created to include the previously mentioned modifier in the compilation script. The default compilation script for the Dymola tool in Linux operating systems is *dsbuild.sh*.

The previously described procedure to parallelize the initial section of Modelica models must be done manually. It could be worth studying how this procedure could be performed directly in the Modelica code and not in the resulting C source code. However, this is not easy, only constant and parameter values can be paralellized with this approach. Obviously, time consuming operation must be involved in the computation of these values because in other case no performance improvement in simulation will be obtained. It must be also taken into consideration the dependencies between these values to properly distribute their calculation between the different threads.

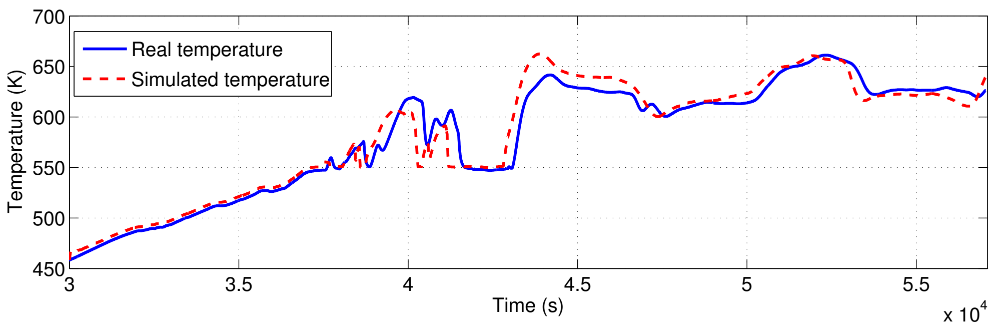

Figure 6. Real and simulated output DISS field temperature in the calibration process.

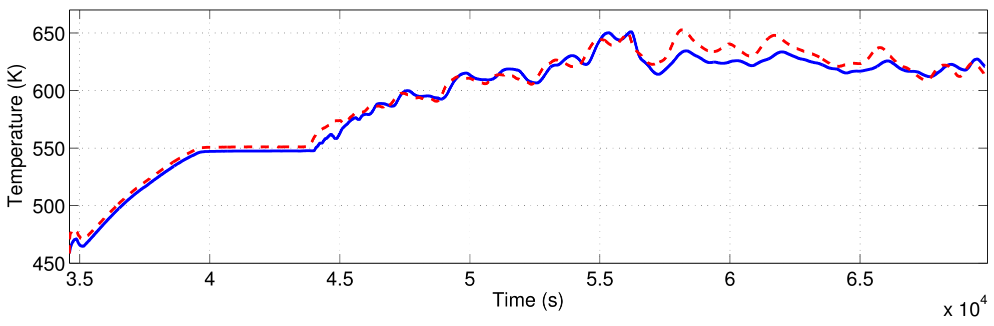

Figure 7. Real and simulated output DISS field temperature in the validation process.

5. Simulation Results

The tests presented in this section were performed using a Intel® Core™ i5 CPU M540, 2.53 GHz. Several tests using different input trajectory files from several operation days were performed and a mean execution time has been calculated. It is important to note that the parallelization is only considered in the initial section and not during the numerical integration.

The simulation statistics are summarized in Table 1. Although the gain in speed is not enormous, just a global speedup of 1.13 was obtained, the calibration process using genetic algorithms is time consuming and even a modest performance improvement is worthwhile. Moreover, if it is considered that the parallelization was only performed in the initial section and not in the whole simulation, the speedup obtained is considerably high, 2.80, because the mean initial section secuential and parallel execution times are 31.231 and 11.127 seconds respectively.

With respect to the calibration process using genetic algorithms, if 20 elements in the population and 100 iterations are considered, the calibration execution time is reduced in 4.5 hours. Although it keeps being time consuming, because it lasted 3 days and 4.5 hours in being performed.

As an example, Figures 6 and 7 show the real DISS test field output temperature (solid line) and the simulated DISS test field output temperature (dashed line) for two different operation days. Figure 6 corresponds to a day used in the calibration process whereas Figure 7 corresponds to a day used in the validation process. The x axis represents the time in seconds from the beginning of the day and the y axis shows the real and simulated DISS test field output temperatures in kelvin.

6. Conclusions and Future Work

The most remarkable conclusions are the following.

- OpenMP can be easily used to parallelize the initial section in the resulting C source code obtained from Modelica models when necessary.
- To take advantage of the parallelization in the initial section of Modelica models it is required time consuming calculations in the initial section.
- The proposed approach has been tested in a dynamic model, which has been validated by experimental data, obtaining a mean speedup of 1.13 in the whole simulation and a mean speedup of 2.80 in the initial section where the parallelization takes place.
- The gain in speed is worth not only in simulation but also in the calibration process where the simulation time is a critical aspect.

Kind of model	Original	Parallelized
Execution time (s)	170.727	151.415
Execution time speedup	1	1.13
Initialization section time (s)	31.231	11.127
Initialization section speedup	1	2.80
Calibration execution time	3 d 9 h	3 d 4.5 h

Table 1. Simulation statistics

As future work it would be useful to study and tackle the following open issues.

- Study how to include mechanisms to describe the use of the OpenMP API directly in the Modelica code instead of using it in the resulting C source code.
- Study how to take advantage of the OpenMP API during the whole simulation, specially in the numerical integration process, and not only in the initial section of Modelica models.
- Take advantage of a 13-node cluster and consider the parallelization not only in the simulation but also in the genetic algorithm calibration method by evaluating concurrently members of the population in each cluster's node.

Acknowledgments

This work has been financed by CIEMAT research centre and UNED own funds, and by the National Plan Project DPI2010-21589-C05-04 of the Spanish Ministry of Science and Innovation and FEDER funds. This support is gratefully acknowledged by the authors. This work has been also performed within the scope of the collaboration agreement between the Automatic Control Group of CIEMAT at Plataforma Solar de Almería, and the Computer Science and Automatic Control Department of UNED.

References

[1] OpenMP, wikipedia the free encyclopedia, `http://en.wikipedia.org/wiki/OpenMP`.

[2] The OpenMP Architecture Review Board ARB. The OpenMP API specification for parallel programming, `http://openmp.org/`.

[3] Peter Aronsson. *Automatic Parallelization of Equation-Based Simulation Programs*. Doctoral thesis No 1022, Linköping University, Department of Computer and Information Science, 2006.

[4] Peter Aronsson and Peter Fritzson. A task merging technique for parallelization of modelica models. In *Modelica conference, Hamburg, Germany*, 2005.

[5] F.E. Cellier. Object-oriented modeling: Means for dealing with system complexity. In *Proc. 15th Benelux Meeting on Systems and Control*, pages 53–64, Mierlo, The Netherlands, 1996.

[6] Andrew Chipperfield, Peter Fleming, Hartmut Pohlheim, and Carlos Fonseca. Genetic algorithm toolbox for use with matlab. Technical report, 1994.

[7] A.B. Dynasim. *Dymola 6.0 User Manual*, 2006. `http://www.dynasim.se`.

[8] J. Eborn. *On Model Libraries for Thermo-hydraulic Applications*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, March 2001.

[9] K. Hongesombut, Y. Mitani, and K. Tsuji. An incorporated use of genetic algorithm and a modelica library for simultaneous tuning of power system stabilizers. In Modelica Association, editor, *Modelica Conference 2002 Proceedings*, pages 89–98, Germany, March 18-19 2002. Modelica Association and Deutsches Zentrum für Luft- und Raumfahrt (DLR).

[10] IAPWS. Release on the IAPWS Industrial Formulation 1997 for the Termodynamic Properties of Water and Steam. Technical report, The International Association for the Properties of Water and Steam, Erlangen, Germany, September 1997.

[11] C.F. Kutshcer, R.L. Davenport, D.A. Dougherty, R.C. Gee, P.M. Masterson, and E. Kenneth. Design approaches for solar industrial process-heat system. Tech. rep. seri/tr-253-1356, Solar Energy Research Institute, Golden (Colorado), USA, 1982.

[12] Håkan Lundvall. *Automatic Parallelization using Pipelining for Equation-Based Simulation Languages*. Licentiate thesis No 1381, Linköping University, Department of Computer and Information Science, 2008.

[13] Modelica Association. Modelica specification 2.2.1, 2007.

[14] Modelica Association. Modelica standard library 2.2.1, 2007.

[15] The MathWorks, Inc. *MATLAB Documentation*. The MathWorks, Inc., 2009.

[16] Hubertus Tummescheit, Jonas Eborn, and Falko Wagner. Development of a modelica base library for modeling of thermo-hydraulic systems. In Modelica Association, editor, *Modelica 2000 Workshop Proceedings*, Lund, October 2000.

[17] L. J. Yebra. *Object-Oriented Modelling of Parabolic-Trough Collectors with Modelica (in Spanish)*. PhD thesis, Universidad Nacional de Educación a Distancia (UNED), Madrid, Spain, 2006.

[18] E. Zarza. *The Direct Steam Generation with Parabolic Collectors. The DISS project (in Spanish)*. PhD thesis, Escuela Superior de Ingenieros Industriales de Sevilla, Seville, Spain, Nov 2000.

[19] Per Östlund. Simulation of Modelica Models on the CUDA Architecture. Master's thesis, Linköping University, Department of Computer and Information Science, 2009.