Parallel Simulation with Transmission Lines in Modelica

Kaj Nyström Peter Fritzson
Dept. of Computer and Information Science, Linköping University
SE-581 83 Linköping, Sweden
kajny@ida.liu.se petfr@ida.liu.se

Abstract

Parallelization of simulations has traditionally been an important way of improving the performance of complex simulations. However, this often requires knowledge in parallel programming, something few modellers have. In this paper we present a way of parallelizing Modelica simulations at the component level requiring no prior knowledge in parallel programming. Our method of parallelizing simulations uses the equation based and unconditionally stable Transmission Line Modeling technique which uses simple time delays to decouple a model into submodels. The method is independent of compiler implementation and thus supports all of the Modelica language supported by a given Modelica compiler. An evaluation of our implementation of this method shows speedups of up to 2.3 times with a variation in speedup that is highly dependent on the model structure and how successful the users parallelization is.

Keywords: TLM; parallel; simulation; transmission line; modeling;

1 Introduction

As knowledge in modeling and simulation becomes more common throughout both industry and academia, the need to simulate systems with higher complexity grows stronger. However, computational power effectively sets the upper limit for how complex our models can be before simplifications have to be made in order for the simulation to finish in reasonable time.

Traditionally one of the most common ways of achieving better performance from a simulation has been to parallelize it. While this is usually difficult for a simulation in a low level language, for example Fortran or C, it is even harder in the Modelica language [4, 5, 6] since the Modelica user has little control over the inner workings of a simulation, something that is often necessary in order to parallelize it. In addition, parallelization of a simulation almost always requires expert knowledge in the area of parallel programming, something that few Modelica users have.

One possibility to simulate in parallel would be to use parallel solvers. These parallel solvers are however not suitable for all problems and can sometimes suffer from numerical instability. Another solution is to automatically parallelize the simulation, either at the Modelica level or at the generated code level. This typically gives better performance for some tightly coupled simulations. However, available tools can not handle for example hybrid models and performance increase could possibly be better if the user can help the modeling environment with the parallelization in some way.

The problem with user interaction when parallelizing a model in Modelica is that user interaction has to be done on a level that the user can access and understand. The typical Modelica user works on the component level. Thus, this is where the parallelization should be specified. It should also be in an application domain neutral fashion, since that is how the Modelica core language is intended to be used. Parallelizing a mechanical application should ideally be no different from parallelizing an electrical application.

2 Contributions

In this paper, we present a domain neutral and numerically stable method of parallelizing Modelica simulations at the component level requiring virtually no knowledge in parallel programming from the user. We base our method on the Transmission Line Modeling theory.

3 Transmission Line Modeling

Central to the task of parallelizing a model is the task of partitioning the model into submodels which can then be simulated on separate computers. We have chosen the Transmission Line Modeling method for parallelizing simulations for a number of reasons. It is a proven way of decoupling equation systems and is also equation based itself which makes it fit nicely into a Modelica component. Furthermore, the TLM method has been proven to be unconditionally stable, an almost absolute requirement as an unstable simulation can be close to useless. This stability holds for as long as the TLM parameters are withing physical boundaries.

The theory evolved from the Telegraphers Equations [18] which concerns signal propagation in cables. In the 1970's the TLM method was first used for computer based modeling by among others A. Fettweiss[11] and P.B. Johns[12]. The method has previously been used to decouple and solve previously unsolvable problems, for co-simulation and to some extent also for distributed simulation.

3.1 TLM Theory

The idea behind the TLM technique is to use physically motivated delays in signal propagation media to decouple a simulation. This time, called T_{tlm} is the time it takes for signals from system 1 to reach system 2 (see figure 1).

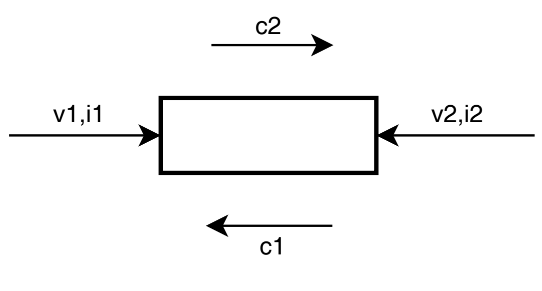

Figure 1: A TLM connection and the governing variables. Notation is from the electrical domain, voltages v_1, v_2 and currents i_1, i_2. c_1 and c_2 are the characteristics of the transmission line .

T_{tlm} can be computed from the signal propagation speed in the medium and the medium length. The equations which govern the exchange of information between the systems are

$$c_1(t) = V_2(t - T_{tlm}) + Z_F I_2(t - T_{tlm}) \quad (1)$$
$$c_2(t) = V_1(t - T_{tlm}) + Z_F I_1(t - T_{tlm}) \quad (2)$$
$$P_1(t) = Z_F I_1(t) + c_1(t) \quad (3)$$
$$P_2(t) = Z_F I_2(t) + c_2(t) \quad (4)$$

The parameters c_1 and c_2 are called the characteristics of the transmission line and represents the propagated information in every time step, delayed as the theory prescribes. P and Q are the variables in the TLM connection and could be from any domain, for example current and voltage. Z_F is an implicit impedance for the connection.

The advantage with introducing TLM connection is that the previously implicit parallelization problem now becomes explicit. Consider the equations 1 and 3. These equations state that P_1 at time t only depends on system 1 and on previous $(t - T_{tlm})$ values from system 2. Thus we have transformed the problem of solving one implicit equation system into two smaller implicit equation systems. The possibility for parallel processing is obvious.

An additional advantage is that we can use both different solvers and different time steps in the two subsystems, as long as we interpolate propagated values reasonably well if needed. This means that we can greatly reduce the stiffness for some problems.

The TLM method has been proved to be unconditionally stable [14, 16], provided that the TLM parameters are computed correctly. The method does not introduce any additional numerical error into the model. Instead, it actually transforms a numerical error to a modeling error[13]. This often makes it easier for a user to identify and compensate for the error rather than if the error would be purely numerical due to the fact that the normal Modelica user is probably a modeling expert rather than a numerical expert.

3.2 Theory Extensions

The TLM theory prescribes that we compute the transmission line delay time T_{tlm} from the propagation speed and the length of the line. However, in order to maximize the degree of decoupling and achieve maximum speedup, we can allow for non-physical T_{tlm}. This is useful since we do not want any stalling in the simulation of the subsystems due to lack of data. This stalling can happen if it takes too long for the data to be transmitted through the computer network from the computer which simulates system 1 to the computer simulating system 2. If we increase T_{tlm} to a value greater than its corresponding computed value, we allow for higher latency which will avoid stalling computations. Such an increase in T_{tlm} is however not without problems. If we increase T_{tlm} too much, the system might become unstable and/or produce wrong results. It is not easy to give an answer on what a good value for T_{tlm} is when you move beyond the strictly physical value as choosing a good T_{tlm} is a trade off between performance and robustness and depends on system dynamics.

The TLM theory was originally developed for electromagnetical signals. Over the years it has been used for a wide variety of domains (hydraulics, mechanics etc). However, we see no reason to limit ourselves to any specific domains since the Modelica language gives us such exceptional possibilities for building generic components.

This method of parallelization should work for most domains in one dimension which propagate one flow and one non-flow variable. Extending the TLM theory to use vectors has been investigated previously[16, 13] in a somewhat different context and we foresee no problem with extending our method to handle different sets of propagated variables.

Since the transmission line has an undamped resonance, it is sometimes beneficial [15] to low pass filter c_i in a transmission line as

$$c_i(t) = \alpha c_i(t-\delta)$$

where α is the filter parameter (0.2 is usually a good value) and δ is the time step. Without this filtering, the resulting signals might contain unwanted high frequency components, resulting in a slightly staircase shaped signal as can be seen in figure 6

4 Implementation

We have implemented and tested our way of parallelization of Modelica simulations with the TLM method as outlined in section 2. The framework consists of 3 parts which we shall now describe in detail.

4.1 A Generic Modelica TLM package

The TLM package (depicted in figure 2) contains the TLM components which the user inserts into his model when he wants to partition it. More on how this is done in section 5.1. The package also contains external functions which take care of the message passing in the simulation. However, the user never needs to see or use these functions.

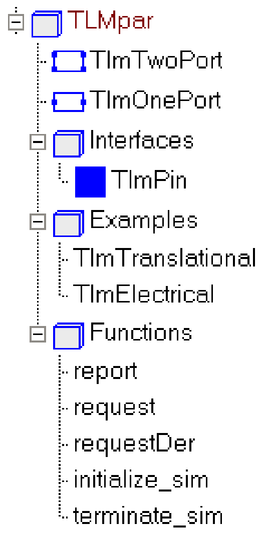

Figure 2: Structure of the TLM package

4.2 The Model Partitioner

This small program transforms the original model into a new Modelica package which in turn consists of the separate submodels derived from the original model. The program also divides and propagates the TLM components so that the correct parts of it are present in all submodels. This partitioning is depicted in figure 3.

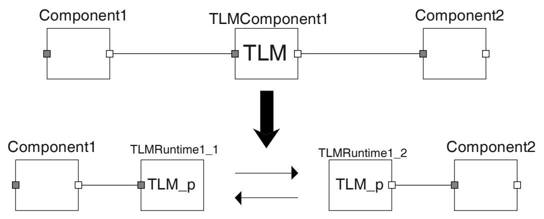

Figure 3: Splitting one model into two models on TLM boundaries

The arrows between $TLM1\_1$ and $TLM1\_2$ symbolize the communication of TLM variables between the two submodels, which takes place over a local network using a simple sockets-based protocol.

Algorithm 1: Partition a model into submodels

```
   Input: An arbitrary component based model
   Output: A mapping component-submodel
           number for all components
 1 $subModelNr \leftarrow 0$
 2 foreach $component \in components$ do
 3     if $notvisited(component)$ then
 4         $visited(component) \leftarrow true$
 5         $component(component) \leftarrow subModelNr$
 6         push(componentStack,
           allNeighbours(component))
 7         while not empty(componentStack) do
 8             $component \leftarrow pop(componentStack)$
 9             $visited(component) \leftarrow true$
10             $component(component) \leftarrow$
               $subModelNr$
11             push(componentStack,
               allNeighbours(component))
12         end
13     end
14     $subModelNr \leftarrow subModelNr + 1$
15 end
```

The partitioning is done using a repeated breadth-first search with visitor recognition as described in algorithm 1. TLM-components and associated connect equations are filtered out before the algorithm is applied to the model as they should be considered as separators (interfaces) between submodels.

After applying this algorithm, all components have an association to a submodel number. We can now insert all components present in the original model in their respective submodels. Next, we check into which submodel the first component in each connect equation belongs and use this information to add the connect equation in the components. For example, if component `R1` belongs in submodel 2, the connect equation `connect(R1.p,C1.n)` should be entered in submodel 2.

The previously filtered out TLM components are now substituted for runtime TLM components as in figure 3. These runtime TLM components contain all necessary functionality for requesting and reporting information necessary for the simulation to their counterpart TLM runtime component.

4.3 A Simulation Dispatcher and Manager

When the partitioning is done, the simulation is built using any Modelica compiler. We have successfully used OpenModelica[1, 2, 3] and Dymola[17]. As far as the compiler is concerned, it is now compiling two or more completely different models with no association between them, so the compiler itself needs no modification.

This gives us some additional advantages. During compilation of our submodels we can customize the simulation of our different submodels, for example choosing different solvers for different submodels if desired. We can also specify different time steps or fault tolerance levels which if done right can significantly reduce the stiffness in the original model.

Finally, the dispatcher takes care of distributing the jobs on separate machines, such as on a computational grid or a PC cluster, and to manage reports and request for data from the simulations.

5 Discussion and Results

In this section we will present and discuss our results with respect to user interaction, performance and fault tolerance.

5.1 User Interaction

One of our primary goals with the work presented in this paper is to provide a way of parallelizing simulation that the average Modelica user can actually use without too much effort. This means that it should require little change in the way the user builds his model. At the time of writing, no scientific study has been done on the usability of this framework so we will settle for briefly describing what has to be done by the user in order to parallelize his model and let the reader decide whether this is usable or not.

The only additional task the user has to undertake in order to use our framework is to partition his model by inserting TLM elements where he wants to partition his model. This can be done either graphically or textually and works exactly like inserting any normal Modelica component in a model. The hardest part for the user is to determine *where* to insert these TLM elements.

The best and most general advice we can give at a model level is to decouple the model at domain boundaries since these are usually the boundaries between fast and slow subsystems. Decoupling such subsystems combined with using different solver settings can significantly increase performance. Another advice is to partition the model in equally complex parts. This is quite difficult to do at a component level since it may be hard to see at a component diagram level what the complexity is of a certain part of a model.

5.2 Performance

Evaluation of a parallel programming framework is difficult at best. Many factors are involved and the framework designer tends to choose the problems and environments that favour his framework the most. When evaluating our framework for Modelica models, we have found that out largest problem by far is to choose our models. Few sufficiently large models are available to the general public, especially models which can be understood and parallelized by a non-expert in the modeling domain.

The best thing would obviously be to have a set of more or less standardized and independent benchmarking models. Lacking this, we have chosen to build our own models for benchmarking using only Modelica Standard Library components and examples. Given this bias problem bias, it is uncertain if we should really present any figures of speedup at all before our framework has been tested with independently built models. Even so, we choose to present our preliminary findings regarding performance here for what they are.

The framework can handle Modelica models or arbitrary size but as usually is the case with parallelization, little or no speedup or even a performance decrease can be expected when parallelizing small models as the communication overhead then becomes a significant factor in simulation time. Then again, there is probably no need to parallelize small models as these will most likely run just fine on a singe CPU.

Using these models we have registered speedups ranging from 0.5 (negative speedup) to 2.3 depending on the structure of the problem and how we parallelize it. Stiff problems which we can easily decouple will give us the greatest speedup while some homogeneous problems might not be suitable for parallelization at all.

All tests were done on a standard PC-cluster with the following nodes

- OS: Rocks Linux [19]
- CPU: PIII-800Mhz
- Memory: 512MB
- Network: 100Mbit Ethernet

The cluster is quite old fashioned but still demonstrates the general effectiveness of our framework. Is is likely that a more modern cluster with faster nodes and faster network will increase performance as we can then decrease the granularity of our model partitioning.

Our figures have been derived by comparing total simulation time on one node to total simulation time using two to six nodes, depending on the model structure. We wish to stress that we do not rule out the possibility that a modeling expert could achieve better performance as he or she might be better suited to parallelize the models.

5.3 Fault Tolerance

Just as the theory predicts, the error in the models we have tested is well within normal values for numerical simulations as long as the TLM parameters are within physical limits. When we go outside physical values however, we will inevitably introduce an error. How large or significant the error is depends on the model. The modeler is obviously best suited to be the judge of if this is within his fault tolerance limits or not. As we are just working with a simple delay in the time domain it is generally easy for a domain expert to see beforehand what effect an extra time delay will have on his model and if this is acceptable or not.

For comparison on what a way too large T_{tlm}, we present two simulation runs of a standard DC-Motor example with a ramp as a voltage source as depicted in figure 4.

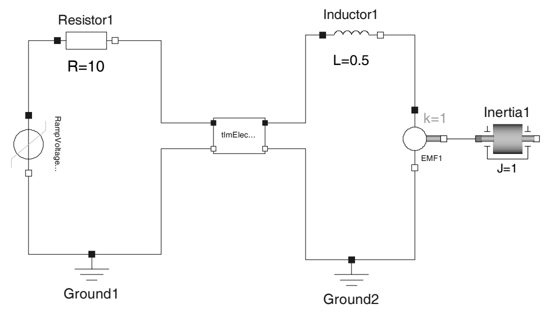

Figure 4: TLM partitioned DC-motor model

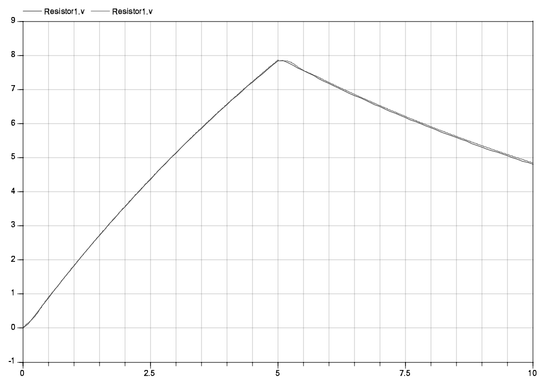

Figure 5: Plot of voltage over the resistor component, T_{tlm}=0.01, interval length=0.01s

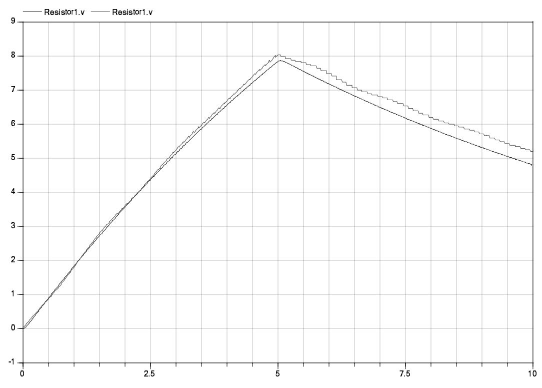

Figure 6: Plot of voltage over the resistor component, T_{tlm}=0.1, interval length=0.01s

From figures 5 and 6, we can clearly see that $T_{tlm} = 0.1$ was probably a to high value for most applications, although it still does show the general shape of the result. Still, since it is a delay in the electrical application domain in a circuit where the propagation speed is usually very large and in a model with no capacitor elements, delaying the signal by one tenth of a second seams rather a lot to any electrical engineer. As always, the modeler must use his judgement when setting the parameter values, in this case T_{tlm}.

6 Conclusions

We have been able to parallelize Modelica simulations and to abstract user interaction at a component level which we believe will be a usability improvement compared to other parallelization techniques. Communication and scheduling are completely hidden from the user. However, no scientific evaluation has yet been done on the usability aspects of our work.

Speedup is up to 2.3 times so far but varies greatly and depends on model structure and if the model is partitioned successfully. Performance also depends on accuracy requirements on the model and is easily configurable by the user. Additional advantages with the method are that it reduces model stiffness if properly used and that it is also possible to use different solvers for different submodels if desired.

7 Future work

Obviously, a usability study is one of the most important items for future work as that has been one of the major goals of our work. We would at the same time like to continue to develop heuristics for better partitioning of models. This process might even be automated using such heuristics. There is also plenty of more work on automatic estimation of non-physical delays that do not lead to errors beyond a given tolerance level, perhaps using static analysis on the model.

A better performance evaluation is also prioritized but largely dependent on the availability of large models. Such models have proven to be difficult to find.

On the implementation side, a better communication implementation pattern (e.g. peer to peer) should be established in order to reduce communication cost. Also, adaptable value reuse depending on model dynamics should not be hard to implement and should lead to a significant decrease in communication overhead.

More static and dynamic analysis of the performance bottlenecks for individual simulations could be a way of aiding the user in both model partitioning and choosing TLM parameters.

8 Acknowledgments

This work was supported by MathCore Engineering[9] and Vinnova[8] in the GRID-Modelica project[10].

References

[1] Peter Fritzson, et al. The Open Source Modelica Project. In Proceedings of The 2nd International Modelica Conference, 18-19 March, 2002. Munich, Germany See also: http://www.ida.liu.se/ projects/OpenModelica.

[2] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. In Simulation News Europe, Issue 44/45, December 2005.

[3] The OpenModelica Users Guide, version 1.3.2, Apr 2006. http://www.ida.liu.se/projects/OpenModelica

[4] The Modelica Association. The Modelica Language Specification Version 2.2, March 2005. http://www.modelica.org. accessed 2005-05-02

[5] Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, 940 pp., ISBN 0-471-471631, Wiley-IEEE Press, 2004.

[6] Michael Tiller. Introduction to Physical Modeling with Modelica. 366 pages. ISBN 0-7923-7367-7, Kluwer Academic Publishers, 2001.

[7] Peter Aronsson and Peter Fritzson, Task Merging and Replication using Graph Rewriting, Tenth International Workshop on Compilers for Parallel Computers, Amsterdam Netherlands, Jan 8-10 2003

[8] Vinnova, http://www.vinnova.se, accessed 2005-05-02

[9] Mathcore Engineering, http://www.mathcore.com, accessed 2005-05-02

[10] The GridModelica Project, http://www.ida.liu.se/labs/pelab/modelica/GridModelica.html, accessed 2005-05-02

[11] A. Fettweiss, Digital Filter Structures Related to Classical Filter Networks. Arch. Elek. Übertragungst., 23(2):79-89, 1971

[12] P.B. Johns and M.A. Brien, Use of the Transmission Line Modeling (t.l.m.) Method to Solve Non-Linear Lumped Networks, The Radio Electron and Engineer, 50:59-70, Jan/Feb 1980

[13] Petter Krus, Modelling of Mechanical Systems Using Rigid Bodies and Transmission Line Joints, ASME journal of Dynamic Systems, Measurements and Control, 1995

[14] S.H. Pulko, A. Mallik, R. Allen, and P.B. Johns. Automatic Timestepping in TLM Routines for the Modelling of Thermal Diffusion Processes. Int. Journal of Numerical Modelling: Electronic Networks, Devices and Fields, 3:127 136, 1990.

[15] P.Krus, A. Jansson, J-O. Palmberg and K. Weddfelt. Distributed simulation of hydromechanical systems. In Third Bath International Fluid Power Workshop, Bath, UK, 1990.

[16] Iakov Nakhimovski, Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis. Ph.D Thesis, Linköping University, Dept. of Computer and Information Science, April 2006.

[17] The Dymola modeling tool, http://www.dynasim.com accessed 2005-05-02

[18] The telegraphers equations, http://en.wikipedia.org/wiki/Transmission_line#Telegrapher.27s_equations, accessed 2005-05-02

[19] Rocks Linux, http://www.rocksclusters.org, accessed 2005-07-20