K. Nyström, P. Aronsson, P. Fritzson
Linköping University, Sweden

Parallelization in Modelica

pp. 169-172

Parallelization in Modelica

Kaj Nyström Peter Aronsson Peter Fritzson
Linköping University, Sweden
{kajny,petar,petfr}@ida.liu.se

Abstract

The better the computer, the larger and more precise simulations can be carried out, and the more benef-icent modeling can be. It is well known that faster computers enable more precise and computationally expensive simulations to be carried out, which allow more pre-cise mathematical models. This paper gives an overview of certain methods for expanding the limits of what can be done in the area of simulation by parallelizing simulations based on Modelica [18, 16] models. This is an efficient and less expensive way of achieving better simula-tion performance.
In the following, we will restrict ourselves to describing various ways of parallelizing a simulation in Modelica, ranging from coarse grained high level parallelization to fine grained task merging at a very low level. It is very difficult to say which approach is the most successful or promising since little research has been done in most of the subareas of parallelizing Mode-lica models. Task merging seems to be the most developed approach and does give significant performance increases [1] but the other areas are largely unexplored. We can therefore only guess that based on parallelization research in other areas, there is little to gain for a normal user in parallelizing a small simulation. Larger, more complex simulations on the other hand can benefit greatly from parallelization, especially if it can be done automatically.
Keywords: Modelica, parallelization, task merging, transmission line modeling, weak connectors

1 Introduction

Since the advent of computers, there has always existed a need for more computational power. In fact, the peak performance of a computer system effectively sets a limit to what the user can actually do. For a Modelica user, the amount of computational power available at simulation time is what determines what can be simulated. Obviously, if we can simulate more complex applications, the use of modeling and simulation is promoted.
There are four different ways which a user can go about in order to faster be able to simulate a more complex physical structure:

1. Buy a faster computer
2. Optimize the model for faster simulation
3. Optimize the compiler
4. Parallelize the model, distributing simulation across many processors.

While option one can sometimes be a viable alternative, it is so only up to a certain point. There is a limit to how fast computers are available, even if you have the financial resources to update your computer every month.
Option two might very well be the issue for a whole series of articles all by itself. Still, experience show that there is a limit to the performance gain which can be obtained also from model optimization.
Option three is an interesting item as it can potentially boost performance of every model compiled with that compiler. The amount of performance gain which can be obtained by compiler optimization is however also quite limited.
This paper will focus on item four, parallelization of the model. The model is assumed to be written in Modelica and to be of a size and complexity which makes it nontrivial to simulate within reasonable time on a single powerful computer. Apart from this, we make no assumptions whatsoever on the model structure, composition or domain.

2 Parallelization in General

Parallelization of computational problems has been an issue for as long as there has been computers. Regardless of how fast current computers are, there has always been applications where this is simply not enough. Parallelization of the problem is then the only viable solution. The problem is to get good performance while distributing the computation across several CPUs. Communication between jobs can, even if carefully implemented cause severe delays in the computation because the necessary data may not be available when it is needed. Also, simple program branches such as if-statements which decide which statement should be executed and not must be treated with great care so that the right program path is taken. These, and several other issues must always be taken into account when trying to parallelize an application. This is even more important when trying to parallelize a general class of applications, such as Modelica models.

3 Parallelization in Modelica

We will in this paper try to summarize past, present and future research advances within the area of parallelization of models expressed in the Modelica language. We have divided the subject into three parts or levels, depending on where the parallelization is applied. The high level parallelization tries to partition the model on the Modelica source code level. The medium level deals with the numerical solvers while the low level parallelization deals with the solver generated code.

Before we begin, we would like to make one observation which makes the task of parallelizing Modelica even more challenging than with other languages, for example C or Fortran. Modelica developers are normally not experts in parallel programming. In fact, they are usually not computer scientists at all, but instead domain experts within one or many specific fields using physical modeling. This demands a lot of the parallelization framework since the user (that is, the Modelica modeler) can not really take an active part in the parallelization. This means that the parallelization must be automatically performed, without user interaction to a great extent. If this is not possible then at least the user interaction must be minimized and formulated in a way that makes sense even to a user with no experience whatsoever in parallel computing.

4 High Level Parallelization

High level parallelization, as stated previously deals with the problem of parallelizing models at the Modelica source code level. In general, this means that the Modelica language itself is extended or modified in some way in order to allow the user to provide the compiler with directions on how to parallelize the simulation. In comparison with other high level languages, the Modelica language has some interesting properties which can be used to our advantage when trying to parallelize Modelica simulations.

The most interesting property is probably the connection construct. A Modelica model almost always consists of a multitude of components with connections between them. The connections define an explicit interface between components which is quite useful when considering how to best partition the model. Indeed, both of the two high level parallelization methods we know about use connections in one way or another.

4.1 The Transmission Line Modeling Method

The transmission line modeling (TLM) method [6] is derived from two ideas. First, that many models can be viewed as a continuous transmission line which propagates information. Second, that the information being propagated in time step $t-1$ in many cases does not differ much from the information propagated in step time step t. This means that we can reuse information received in time step $t-1$ in the calculations for time t, thus reducing the amount of communication needed between partitions in the model. While we do introduce an error in the model by reusing values from earlier time step, this error is mathematically decidable and it is possible to reduce the amount of value reuse and thus reduce the error introduced. The transmission line modeling method is not yet implemented in any Modelica implementation that we know of although an implementation of the TLM method has been planned in the GridModelica project [5] for 2005.

4.2 The Weak Connectors Method

Introduced by [7] this somewhat less explored method also deals with connections. By introducing the concept of weak connections, the model can be partitioned in two or more parts. The idea is to separate fast subsystems from slow so that different solver step size, or even different solvers can be used when solving the system. The difficult part here is to find good places to insert the weak connection, instead of a normal connection. Such places frequently occur between domain boundaries and while these could quite easily be identified by a domain expert, it is not so easy to find them automatically, which is of course the desirable method. One way of doing this could be to exploit the package structure of Modelica which roughly divides components into different domains.

4.3 Other high level parallelization methods

There are a some high level parallelization techniques in traditional parallel programming that could be adapted to Modelica. One such technique is matrix operation partitioning. Matrixes and vectors represent large data chunks upon which operations are executed. One example operation could be to add one to each element in the matrix. Such an operation could quite easily be distributed across several CPUs as the individual operations of adding one to element $m[i][j]$ is independent from the operation of adding one to element $m[k][l]$.

In the same way, parts of normal loop parallelization techniques could probably be employed to achieve parallelization in Modelica. For example High Performance Fortran (HPF)[11] has the *forall*, *pure* and *independent* keywords which gives the compiler directions on how to parallelize loops in the program. Even though these constructs could quite easily be introduced in the Modelica language, it is unsure whether they will provide the same performance boost as they do in HPF due to Modelicas radically different execution model.

5 Medium level parallelization

The next level of parallelization is at the equation system and numerical solver level. Parallel solvers have in the past had problems with numerical stability in comparison with other state-of-the-art solvers. Thus, limiting the usage of such solvers to specific domains where the requirement on the numerical stability of the solver is not too demanding. Parallelizing numerical solvers is in itself a very complex task and while an interesting way to achieve additional parallelism, for example with algorithms such as [12, 13] it is not really Modelica specific. We shall therefore in this paper concentrate on other ways of equation parallelization.

Another interesting solver related technique is the mixed mode integration technique presented in [4]. It is a compromise between explicit and implicit integration, done by splitting fast and slow subsystems in a model and to apply implicit discretization only to the fast part. Results presented in [4] indicates performance increases ranging from four to sixteen times.

One task that could be parallelized without too much effort be parallelized is the Jacobian calculation. Jacobian calculation is sometimes necessary when using an implicit ODE solver and its calculation is side effect free which makes the amount of interCPU communication small. Related to this, it is possible to achieve some degree of parallelism in the calculation of the states in an ODE or a DAE, meaning function f in the ODE system 1 and functions f,g in the DAE system 2.

$$\dot{X} = f(X,t) \tag{1}$$

$$f(\dot{X},X,Y,t) = 0, g(X,Y,Z) = 0 \tag{2}$$

It might also be possible to conduct parallelize solving of equation system in some cases, as done in [14]. Even though it is common that subcomponents in an equation system depend upon each other in a linear fashion, it does not have to be so. What has to be done is to build a task dependency graph and determine if subsystems can be solved simultaneously and pass this to a task scheduler which then distributes the tasks.

Scheduling and partitioning algorithms as described in [2] also belongs on this level. In that paper only static scheduling algorithms are described and while these work very well for continuous systems, they will not work with hybrid models, meaning models that contain both continuous and discrete parts. In such hybrid systems, discrete events can radically change the behavior of the system so in that case, we need to use dynamic scheduling instead.

6 Low level parallelization

While the difference between medium an low level parallelization might be hard to define, we have in this paper drawn the line at the data level at which the parallelization algorithms work. With low level parallelization, the object is to parallelize the compiler generated simulation code. We will refrain from describing low level parallelization here since it is already thorouwghly described in [3].

7 Discussion

Parallelization in Modelica is still very much underdeveloped, with the possible exceptions of the low level parallelization and solver integration. This is perhaps somewhat surprising since physical modeling and simulation is one of the areas with the strongest demand for more computational power. While there exist some parameter study applications [10, 9, 8], real parallelization of Modelica models is still to a great extent a an open issue to be explored.

While the above mentioned techniques can probably be applied separately with good results, even better results can be expected if they are combined together. For instance, parallelization of the model with the TLM method can be combined with task merging in the lower layer to achieve a coarse grained parallelization at Modelica source level while achieving a more fine grained parallelization at lower level.

To conclude, we think that while a modeling language perhaps not does not live or die with its parallelization abilities, it is still important to develop parallelization in order to make the Modelica language a serious competitor to Fortran, C and C++ also when it comes to simulation of computationally demanding models.

8 Acknowledgement

This work was sponsored by Vinnova[19] via the GridModelica[5] project.

References

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

[2] Aronsson P., Fritzson P. *Clustering and Scheduling of Simulation Code Generated from Equation Based Simulation Languages* 9th Workshop on Compilers for Parallel Computers, CPC 2001, June 27-29, Edinburgh, Scotland, UK

[3] Aronsson P. Fritzson P. *A Task Merging Technique for Parallelization of Modelica Models* Modelica conference 2005, Hamburg, Germany 2005.

[4] Schiela A. Olsson H. *Mixed-mode Integration for Real-time Simulation* Proceedings of the Modelica Workshop 2000, pp 69-75.

[5] The GridModelica project, http://www.ida.liu.se/labs/pelab/modelica/ GridModelica.html, accessed 2004-12-06.

[6] Christopoulos C. , The Transmission Line Modeling Method, EEE/OUP Series on Electromagnetic Wave Theory, 1995

[7] Casella F. , Maffezzoni C. : ”Exploiting Weak Interactions in Object Oriented Modeling”, EUROSIM Simulation News Europe, Mar. 1998, pp. 8-10.

[8] Nyström K, Aronsson P, Fritzson P. *GridModelica - A modeling and simulation framework for the grid* Proceedings of SIMS conference, Copenhagen, Denmark 2004.

[9] Engelson V, Fritzson P. *A Distributed Simulation Environment* Proceedings of SIMS 2002 conference.

[10] Joos H-D, Looye G, Moormann D. *Design of Robust Inversion Control Laws using Multi-Objective Optimization* AIAA Guidance and Control Conference, Montreal, Canada, 2001.

[11] High Performance Fortran Forum *High Performance Fortran Language Specification, version 1.0* Houston, Texas, 1993

[12] Yi-Ling F. Chiang, Ji-Suing Ma, Kuo-Lin Hu and Chia-Yo Chang *Parallel multischeme computation* Journal of scientific computing, 3(3):289-306, 1988

[13] Gilbert C. Sih and Edward A. Lee *Declustering: A new multiprocessor scheduling technique* IEEE Transactions on Parallel and Distributed Systems, 4(6):625-637, June 1993

[14] Andersson Niklas *Compilation of Mathematical Models to Parallel code* Licenciate thesis, Linköping University, 1996

[15] *The OpenModelica project.* http://www.ida.liu.se/ pelab/ modelica/OpenModelica.html

[16] *The Modelica Association.* `http://www.modelica.org`.

[17] Fritzson P, Aronsson O, Bunus P, Engelson V, Johansson H, Karström A, Saldamli L. *The Open Source Modelica Project*. In Proceedings of the 2nd International Modelica Conference, Munich Germany, 18-19 2002.

[18] Fritzson P. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.

[19] *Vinnova* `http://www.vinnova.se`