Multi-Threaded Real-Time Simulations of Fluid Power Systems Using Transmission Line Elements

M.Sc. Robert Braun

Division of Fluid and Mechatronic Systems, Linköping University, SE-58183 Linköping, Sweden, E-Mail: robert.braun@liu.se

Professor Petter Krus

Division of Fluid and Mechatronic Systems, Linköping University, SE-58183 Linköping, Sweden, E-Mail: petter.krus@liu.se

Abstract

The demand for large-scale real-time simulations of fluid power systems is increasing, due to growing demands for added functionality. Real-time simulations can be used in for example hardware-in-the-loop experiments and embedded control systems. In order to achieve real-time performance, it is often necessary to use small or simplified models, reducing the usefulness and accuracy of the results. This article proposes the use of transmission line modelling (TLM) for exploiting multi-core hardware in real-time and embedded systems. The characteristics of the TLM method are analysed to identify difficulties and possibilities. A method for how to parallelise TLM models is then presented. Subsequently, a programming interface for implementing the parallel models in the target systems is introduced. Practical experiments show that the approach works and that the method is applicable. So far, however, it has required great effort on the part of the engineer, both when it comes to programming, compiling and importing the model into the target environments, although some attempts to automate the procedure have been successful, reducing the level of complexity.

KEYWORDS: Real-time simulation, Distributed modelling, Transmission line modelling, Parallel simulation, Multi-core, Model fidelity

1 Introduction

Demands on functionality and controllability in fluid power systems are increasing. As a consequence, sensors and embedded software are being introduced in many systems. This opens up new possibilities of real-time simulations, which can be used in for example hardware-in-the-loop experiments and human-in-the-loop simulators. Achieving real-time performance, however, requires much com-

puter power. With present technologies, high fidelity simulation in real-time is not possible for other than rather simple systems. Specialised, highly simplified and often linear models must often be used, resulting in less reliable simulation results. In contrast to this, a common desire is to be able to use the same simulation model throughout the entire product life cycle. This requires very complex high-fidelity models, which are not adapted to real-time simulation. To achieve real-time performance for such models it is necessary to fully exploit all the available computer power. This would enable high-fidelity hardware-in-the-loop experiments of complex systems, which can reduce cost and time requirements in product development. The particular problem studied in this project is a hardware-in-the-loop test rig for developing hybrid drive trains for heavy vehicles. These require a significantly higher level of control than conventional drive trains. High-fidelity models are therefore necessary to develop these control systems.

Previously, it has always been possible to rely on the computer processor speed increasing. Moore's Law states that the number of transistors in integrated circuits will approximately double every year. While this still holds, the increase in processor speed has however flattened out. There are several reasons for this, but it is mainly due to the fact that the frequency in a processor is proportional to the voltage and thereby to the square of the power dissipation. Further frequency increases would be too energy-inefficient and require expensive cooling systems. This has resulted in a trend among manufacturers to build processors with several slower cores, so called multi-core processors. /1/

For running heavy simulation models with real-time performance, exploiting multi-core technology is inevitable. Achieving parallelism in system simulations, for instance models of hydraulic circuits, is however difficult. In most cases, an ordinary differential equation solver (ODE) or a differential algebraic equation solver (DAE) is used to find a numerical solution. These by nature contain many dependencies between different variables, limiting the possibilities for parallelism.

An alternative approach providing interesting advantages is to use a system model that is composed of distributed subsystems or components. Each such component solves its own equations independently using distributed solvers. Using distributed simulations can even be an absolute necessity when working with models from different vendors, in order to limit disclosure of restricted information. Distributed solver approaches however require the parts of the models to

be independent in the time domain. One simulation technique that satisfies these two conditions is the *transmission line element method*, which also happens to be especially suitable for fluid power systems.

The contributions of this paper are as follows:

- The characteristics of the transmission line element method are analysed, to identify the possibilities for parallelism.
- A method for how to divide transmission line element models is proposed.
- A programming interface for implementing multi-threaded transmission line element models in a real-time system is presented.
- Practical experiments are provided to verify conductibility and to measure the effectiveness of the method.

2 Multi-Threaded Transmission Line Modelling

The transmission line element method (TLM) is a modelling method for one-dimensional power transmitting systems. It is related to the method of characteristics, which was first used in the HYTRAN tool to simulate the hydraulic system in the NASA space shuttle /2/, and to transmission line modelling of pressure-flow systems as described in /3/, originally known as bilateral delay-lines /4/. The main difference compared to other simulation methods is that TLM components not only communicate with pressure and flow, but also with characteristic impedances. This makes it possible to simulate wave propagations very accurately. Components are divided into capacitive components (C-type) and resistive components (Q-type). At any given time, one component is never dependent on another of the same type. This means that all C-type components can be simulated in parallel, and subsequently all Q-type components in parallel, with a physically motivated time delay in-between. This is achieved by using distributed solvers, so that all components solve their own equations.

Due to the time independences and the absence of a centralised solver algorithm, TLM models are very suitable for parallel simulations /5/. By distributing the work load of a simulation over several processor cores, simulation performance can be greatly increased. Ideally, the speed-up factor will increase linearly with the number of cores. In reality though, multi-threading programs suffer from overhead time costs, which reduces the benefits. As model size increases the significance of the overhead is however reduced, making the speed-up almost linear to the

number of cores /6/. Equation 1 explains the relationship between relative speedup ($SU_{rel}(p,n)$), execution time with one processor ($t_A(1,n)$) and execution time with p processors ($t_A(p,n)$). The variable p must not always match the actual number of processors, but can more generally be defined as the *degree of parallelization*. The difference between single-threaded and multi-threaded simulations can be illustrated as shown in Fig. 1 and Fig. 2.

$$SU_{rel}(p,n) = \frac{t_A(1,n)}{t_A(p,n)} \tag{1}$$

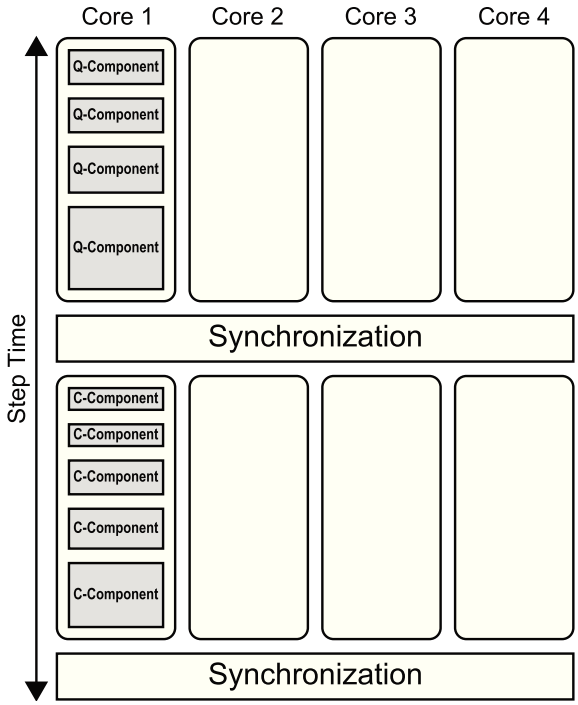

Figure 1: Letting one core do all the work is inefficient.

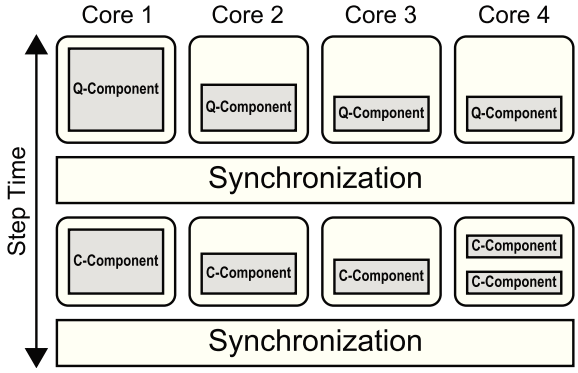

Figure 2: Distributing the work load can reduce simulation time.

Previous attempts to run parallel simulations by using the TLM method have been made using networks of separate computers /7/, and subsequently transputer technology, which was an early predecessor of multi-core processors /8/. An automated algorithm for multi-core support in the Hopsan simulation package was implemented in /6/. Hopsan is an integrated simulation environment for fluid and mechatronic systems based on the transmission line element method. Development first began at Linköping University in the late 1970s and it has been used by industry as well as for research. A new generation of the software was released in 2011. It is the only simulation package that utilises the TLM technology within mechanical engineering and fluid power. /9//10/

The new generation has built-in support for parallel simulations on multi-core processors, by using the Threading Building Blocks from Intel. The simulation core will automatically measure time requirements for each component, and partition models to achieve good load balancing /6/. This function however depends on both external libraries and an operating system with a task scheduler. For this reason it is not suitable for embedded real-time systems, which may only support a single binary file or require manual mapping between threads and processor cores. The solution for such systems is based on the idea that several components of C-type and Q-type can be combined into larger sub-models, also of C-type or Q-type. Each such sub-model can then be compiled into a separate library file, to be executed on separate processor cores.

3 Method of Implementation

The target system used for implementation is a real-time multi-core computer using the Windows XP Embedded operating system. It has a processor with four cores, each capable of simulating with a time step of down to 1 μs. Simulation, communication and data acquisition are handled by a software platform based on LabVIEW and developed by Prevas /11/. Shared library files (.dll) with precompiled code can be uploaded to the target through a graphical interface on the host computer, see Fig. 3. Input ports, output ports and parameters in these can then be routed either to each other or to external sensors and control systems. The libraries must be compiled according to the National Instruments' *Veristand* framework, which is used by the *Simulation Interface Toolkit*. This specifies an API with functions for initialising , simulating and accessing ports, signals and parameters. Parameters are double-buffered, meaning that parameters have one copy for read operations and one for write operations, to enable thread-safe con-

current read and write operations /12/. The code is executed through three main functions called $USER\_initialize()$ which is run once at the beginning of the simulation to initialise the model, $USER\_takeOneStep()$ which is called at every time step and handles the actual simulation computations, and $USER\_finalize()$ which can be used at the end for cleaning up after the simulation.

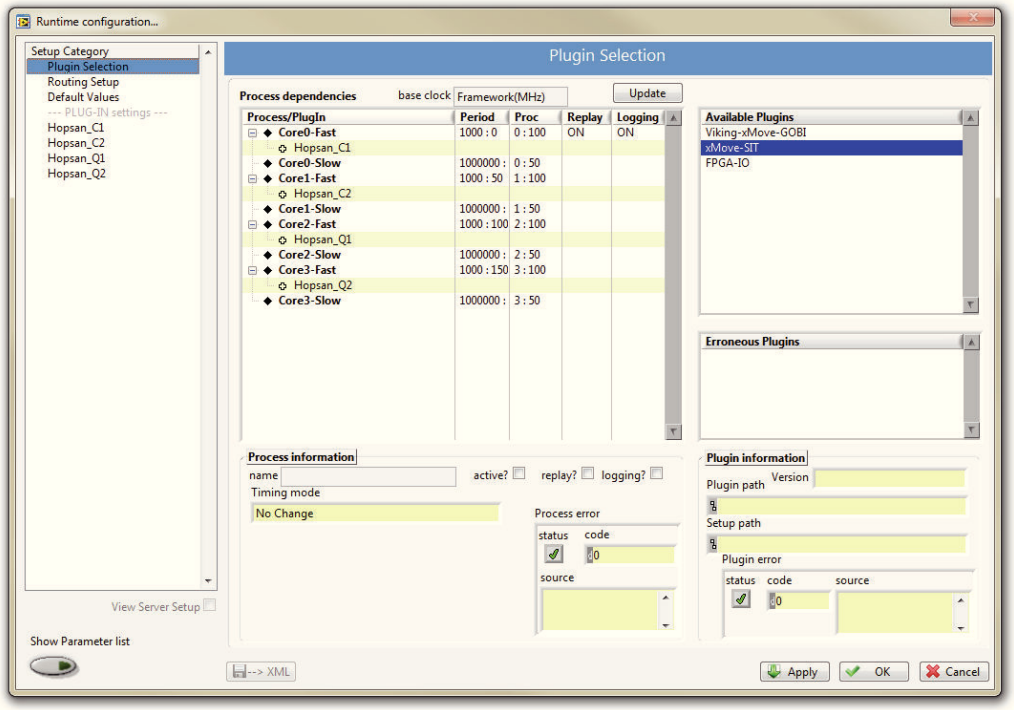

Figure 3: A LabVIEW-based software interface was used to control the real-time target system.

In non-real-time multi-threaded simulations in Hopsan, models are parallelised component by component. First, all C-type components are run in parallel. Then the threads are synchronised, and when all threads are finished the Q-type components are run in parallel. This works well because it is easy to automate and requires no manual adjustment of the model /6/. It is not, however, a feasible method for real-time simulations, since it requires a great deal of communication between different parts of the model, which causes time delays and must be manually routed. This is not the case in non-real-time simulation because a shared memory can be accessed directly from each component. In real-time, each part of the model also requires its own pre-compiled library. Too many libraries will inevitably cause additional overhead costs. An alternative approach is to partition the model into several larger sub-models, also of either C-type or Q-type, see Fig. 4. Ideally, there should then be one such sub-model on each core,

or fewer if the system is small, to reduce further overhead costs.

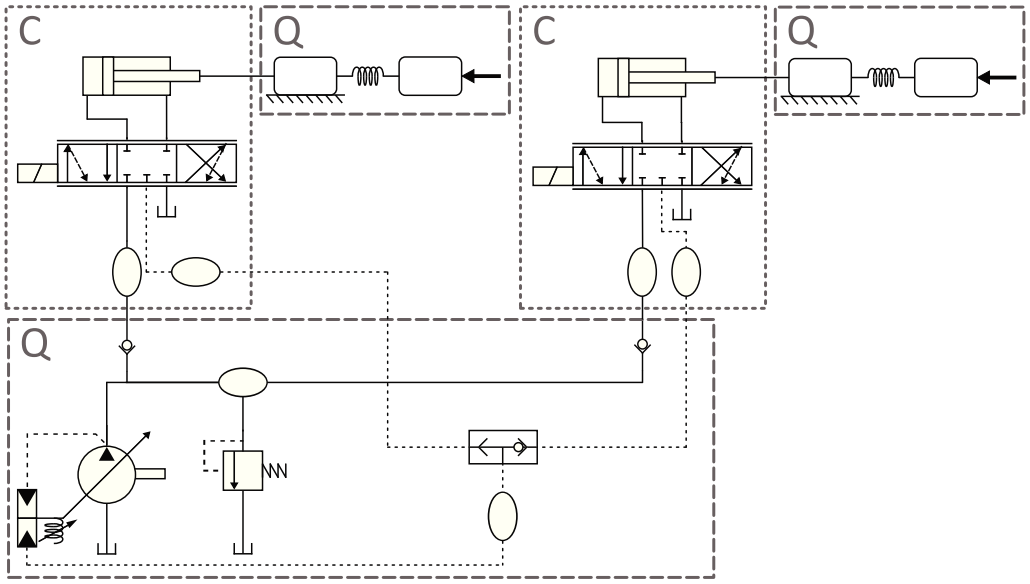

Figure 4: With transmission line modelling, systems can be divided into time independent subsystems of C-type or Q-type.

Each of these sub-models must be compiled in to a shared library together with the simulation core. This is due to the fact that the real-time system in its current state cannot handle libraries that depend on other libraries. For similar reasons, the models cannot be loaded from a model file, but must instead be hard-coded into the library by communicating directly with the simulation core API. Doing so directly would obviously result in complicated coding. To increase the level of abstraction a set of wrapper functions was created, see Fig. 5. These make it possible to initialise a model, add components, define connections, set parameter values, initialise the simulation and simulate one step at a time.

Ideally, C-type and Q-type components shall never be ran simultaneously, due to the fact that a component is only independent of other components of its own type. The physically motivated time delay between components should be preserved if possible. This can be achieved in two ways. The easiest method is to only execute the code from each library every second time step, with C-libraries on odd steps and Q-libraries on even steps. A more complicated approach would be to include two systems in each library, one of each type, and alternate between these every other time step. A third possibility might be to use the first method, but assign two libraries to each core. Another possibility would of course be to

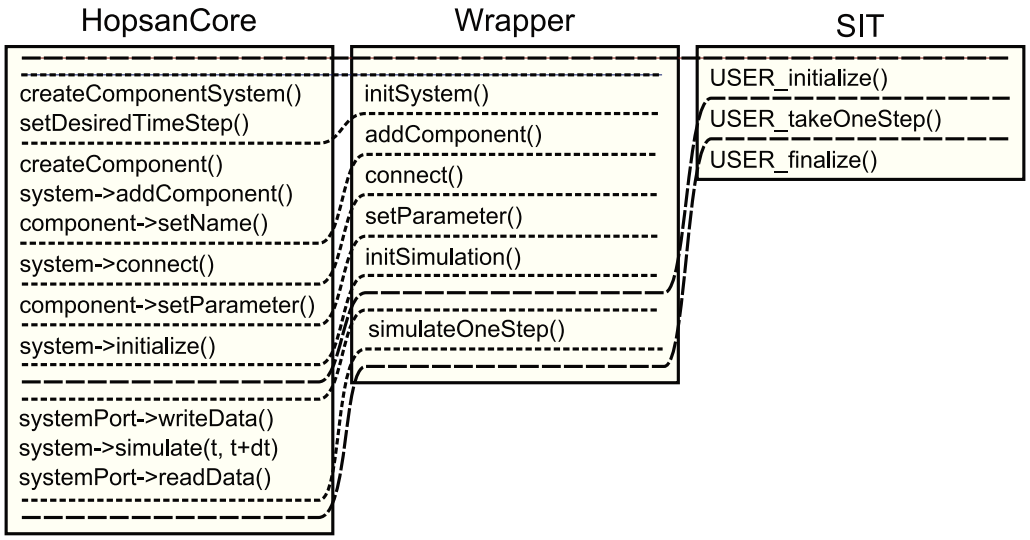

Figure 5: Wrapper functions are used to increase the level of abstraction in the library code.

investigate whether the impact of not taking the time delays into account really is significant, by running all subsystems simultaneously and always using the most recent node variables. This would likely give erroneous results for wave propagations, but if the overall effect is small it may still be useful for simulations where wave phenomena are of less importance. Only the first and third approaches have been considered in this paper.

The Hopsan simulation core requires all used ports in a model to be connected. For this reason, it is necessary to place special interface components at the boundary ports on each system, see Fig. 6. These components contain no equations, but add the necessary boundary connections and allow the wrapper library to access variables in the node with the `readNode()` and `writeNode()` commands. The actual communication between subsystems is then handled externally by the simulation platform in the target system according to the signal routing specified in the user interface. These connections can also involve other signals in the system, such as libraries from other sources, measurement signals from sensors or control signals. This enables powerful tools such as co-simulation and different kinds of in-the-loop simulations. In this article only Hopsan libraries are considered.

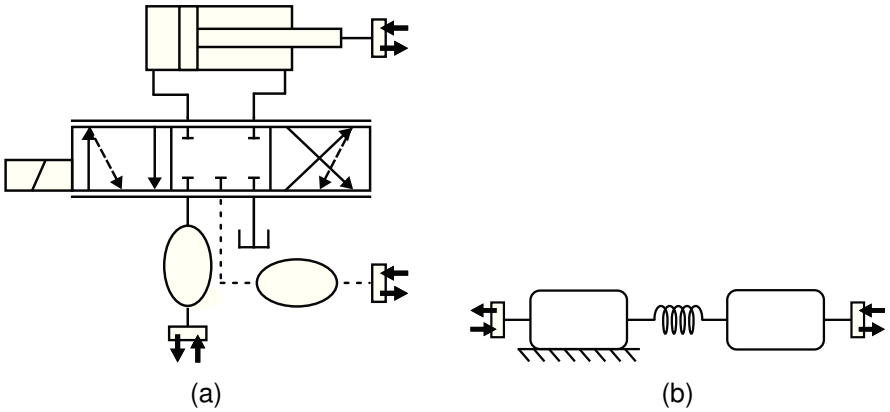

Figure 6: Interface components, shown as rectangles with black arrows, must be used at the boundary ports of the sub-models.

4 Analysis

The performance of the target system was measured by using a benchmark library, which calculates the faculty of its input signal. The number of arithmetic operations will thus be proportional to the input signal, making it possible to adjust the computational load during the simulation. When the computations on a processor core take too long, it reports this as a missed iteration. The computational power could be measured by executing isolated benchmark libraries on different cores, increasing the computational load and observing at which level it begins to miss iterations. The maximum computational load in each core was measured to be roughly $185\,\mathrm{MFLOPS}$. This corresponds to a load sensing system similar to figure 4 with approximately 230 cylinders and a time step of 1 ms. When the benchmarking code was wrapped into a Hopsan component, which should theoretically induce some additional overhead, no significant reduction of this value could be seen. It should thus be possible to run models the size of a load sensing system with more than 900 cylinders when using all processors. In a few of the measurements the performance was found to be significantly lower. No explanations for this have been found at the present time.

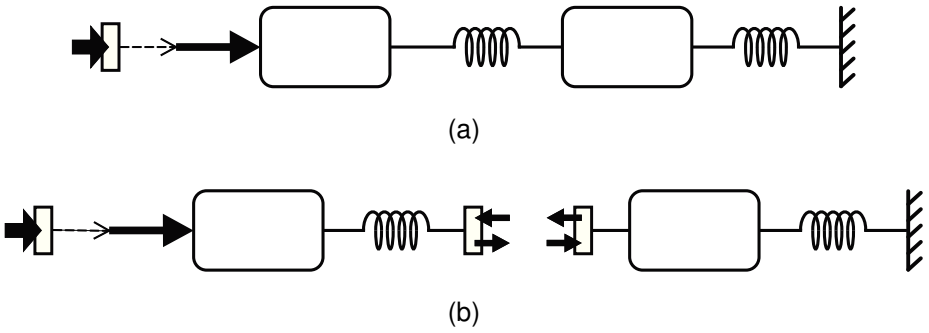

Figure 7: A double mass-spring model was used for verification. It was separated into a C-type and a Q-type sub-model.

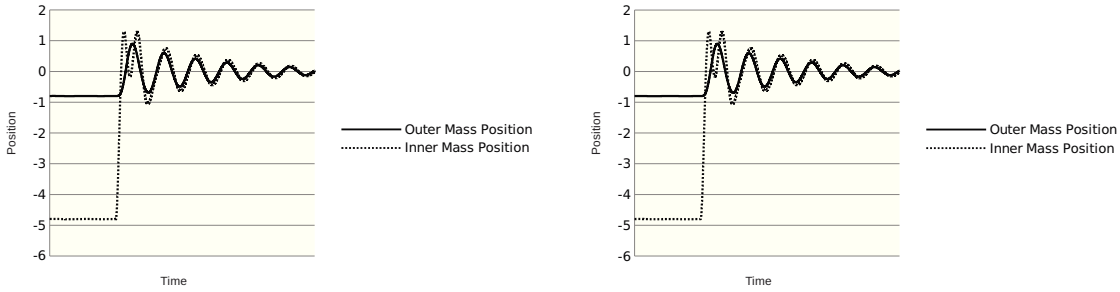

Figure 8: Step response of the mass-spring system in Fig. 7 with single-threaded simulation (a) and multi-threaded simulation (b).

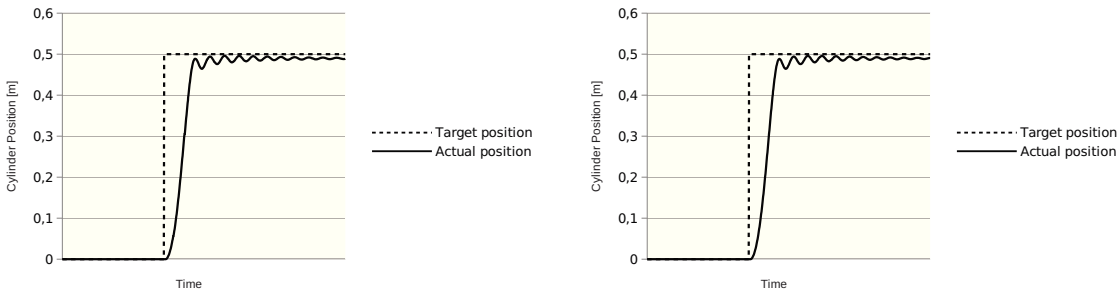

Figure 9: Step response of a hydraulic position servo system with single-threaded simulation (a) and multi-threaded simulation (b).

5 Conclusions & Future Work

Achieving parallelism in real-time systems by dividing models with transmission line elements is feasible. Results show no noticeable effects on numerical properties or simulation results. The communication time delay in the transmission lines can be maintained by running C-type and Q-type sub-models every odd and

even time step, respectively. This means that each model is only called every second time sample in the host system. The host system sample frequency must therefore be twice of the model frequency, in order to maintain the correct simulation time step. Total speed-up is not reduced by this method because several sub-models can be assigned to the same core.

The system used in these experiments was capable of processing very large models without losing real-time performance even in single-threaded simulations. However, this is not always the case and the methods derived in this paper should be applicable on slower computers as well. The experiments showed that neither wrapping the simulation core into dynamic libraries nor introducing the additional communications necessary for multi-threading caused any significant overhead time costs. This percentage, however, is likely to increase on slower machines, possibly reducing the benefits.

To achieve maximum benefit from multi-threading, the model must be divided in such a way that the total communication between the sub-models is minimised. This also facilitates the implementation on the target system. The limitation of only being able to use one dynamic library file for each sub-model makes the process cumbersome. The source code for the simulation engine must be available and the model must be hard-coded into the library.

A continuation of this work might be to introduce support for using additional dependency files. This might for example be pre-compiled libraries, which are necessary to protect source code in propriatory software. It would also be very useful to be able to load models from external model description files. Another continuation might be to verify the results on smaller embedded systems with more limited computation capabilities. In the long run, the goal is to run hardware-in-the-loop simulations with large-scale high-fidelity models and small time steps.

References

/1/ H. Sutter. A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*.

/2/ Air Force Aero Propulsion Laboratory. Aircraft hydraulic system dynamic analysis. Technical report, Air Force Aero Propulsion Laboratory, 1977.

/3/ P.B. Johns and M.A. O'Brian. Use of the transmission line modelling (T.L.M)

method to solve nonlinear lumped networks. *The Radio And Electronic Engineer*.

/4/ D.M. Auslander. Distributed system simulation with bilateral delay-line models. *Journal of Basic Engineering*, pages 195–200, June 1968.

/5/ P. Krus, A. Jansson, and J.O. Palmberg. Distributed simulation of hydromechanical system. In *The Third Bath International Fluid Power Workshop*.

/6/ R. Braun, P. Nordin, B. Eriksson, and P. Krus. High performance system simulation using multiple processor cores. In *The Twelfth Scandinavian International Conference On Fluid Power*, Tampere, Finland, May.

/7/ A. Jansson and P. Krus. Real-time simulation using parallel processing. In *The Second Tampere International Conference On Fluid Power*.

/8/ J.D. Burton, K.A. Edge, and C.R. Burrows. Partitioned simulation of hydraulic systems using transmission-line modelling. *ASME WAM*.

/9/ M. Axin, R. Braun, A. Dell'Amico, B. Eriksson, P. Nordin, K. Pettersson, I. Staack, and P. Krus. Next generation simulation software using transmission line elements. In *Fluid Power and Motion Control*, Bath, England, October.

/10/ B. Eriksson, P. Nordin, and P. Krus. Hopsan NG, a C++ implementation using the TLM simulation technique. In *The 51st Conference On Simulation And Modelling*.

/11/ Prevas. *Viking GUI - User Manual*, October 2010.

/12/ National Instruments. *Using the NI Veristand^{TM}Model Framework*. National Instruments Corporation, Austin, USA, latest edition, 2009. See also URL `http://www.ni.com/pdf/manuals/372952a.pdf`.