

SEMI-AUTOMATIC PARALLELIZATION OF SIMULATIONS WITH MODEL TRANSFORMATION TECHNIQUES

B. Kaan Görür
Roketsan A.S.

PO Box 30, Elmadag, Ankara, Turkey
kaan.gorur@roketan.com.tr

A. Nebi Çallı
Roketsan A.S.

PO Box 30, Elmadag, Ankara, Turkey
nebi.calli@roketan.com.tr

ABSTRACT

The popularity of parallel and distributed simulations (PADS) is increasing gradually, because they are one of the strongest techniques to simulate complex models in a scalable way. Thanks to PADS, high speedup values can be achieved and many of those complex models can be simulated in acceptable time periods. Despite this advantage, most of the engineering simulation developers cannot allocate their time to parallelize simulations, due to the limitation of development time. Moreover, employing PADS requires developers to have important amount of knowledge in parallel software development; but model developers who are not from software engineering discipline generally do not have to know how to develop PADS. On the other hand, running simulations in a non-parallel way causes enough number of experiments cannot be done because of the tight project deadlines. In this paper, a user-assisted parallelization method is proposed for simulations that have been implemented using model-based development approach.

Keywords: Parallel and distributed simulation; model transformation; model-based development; block diagrams.

1 INTRODUCTION

Simulation of complex behaviors can be run in a scalable manner by employing parallel and distributed processing. Many of those complex models can be simulated in acceptable time periods by employing PADS techniques. There are two main reasons to make simulations parallel or distributed. The first and the most well-known reason is the necessity of faster computation (Fujimoto 2016). To simulate complex models in shorter time periods, multiple computation resources can be brought together to share jobs in the simulation. Therefore, independent parts of a model can be solved by multi processors and merged in one of those processors. Sometimes, millions of runs should be executed to discover more accurate results, such as Monte Carlo simulations. If the project deadline is tight and there is no time to run simulation with various conditions, then project teams have to skip many experiments. This situation causes that more accurate results may be missed (Kalos and Whitlock 2008). The second reason is much more about interoperability and reusability (Fujimoto 2016), that are two of the most studied topics in modeling and simulation community, rather than performance consideration. Model owners would like to run their simulations together with other researchers' or foundations' simulations without sharing their own models. Therefore, many distributed computing techniques and standards have been developed for last few decades.

Most of the engineering simulations are designed and developed via advanced mathematical and engineering tools, such as Matlab, Scilab and Octave. In parallel with the developments in model-driven

engineering, model-based development extensions of those tools have appeared in the last decade. Matlab/Simulink, Scilab/Xcos (Xcos 2016) and such kind of tools have been accepted by developers, because they make the simulations more understandable, and easy to develop and modify. Simulink and Xcos provide an interface to developers for designing simulations via block diagram that is one of the most popular model-driven design tools for simulating complex engineering problems. Those tools are most demanding engineering simulation software in many fields, including automotive, aerospace and defense industries.

Despite the advantages of PADS, developing them requires a special care. Engineering simulations are mostly developed by people who are from various kind of disciplines. They are not expected to know about the principles of parallel and distributed computing. When those simulations are heavy for a single processor, making them parallel or distributed appears as a too difficult problem for engineers who are not software experts.

In this paper, we propose an approach to parallelize simulations, that are developed with block diagrams, by leveraging model transformation techniques. We also present a case study that parallelize a simulation model that is developed in Scilab/Xcos that is an open source alternative to Matlab/Simulink. We describe our approach as semi-automatic, because the developer should give a clue about parallelization. The main contribution of this approach is giving a parallelization facility to developers who needs to run a parallel simulation but do not know so much about PADS. Using our approach, developers, not only from software engineering and computer science, can parallelize their simulations and efficiently use the computation resources they have.

2 RELATED WORK

In the last decade, software development methods have evolved with the appearance of model-driven development tools. Those tools provide an abstraction layer to conventional programming languages to manage the complexity (Hutchinson, Whittle, and Rouncefield 2014). Simulink and Xcos are such kind of visual development tools that are used by many engineering software developers. On the other hand, parallel and distributed computing techniques are widely used in the modeling and simulation community to take advantage of using multiple processors for simulating complex problems that take too much time of a single processor. To this end, researches in the last decade focuses on parallelizing models that have been designed with model-driven design tools. Ozard and Desira's (2000) study is the first one that proposes an automated approach for distributing Simulink software to multiple CPUs to reduce execution time. They target the models that contain natural parallelism which parallel subsystems do not interact with each other except at the end of a time step. Task of a parallel subsystem in that study is making some computations and gathering simulation results in a different node in the simulation.

Canedo, Yoshizawa and Komatsu (2010a) proposed a compilation structure, namely strands, to break data dependencies and create a concurrent representation of the program without damaging the original semantics of the model. In another study of them, a novel skewed pipelining technique that allows loop-carried Simulink applications to be executed concurrently in multi-core systems has been proposed (Canedo, Yoshizawa and Komatsu 2010b). A performance improvement has been observed by loosening the dependency among simulation steps. The main drawback of that study is the trade-off between accuracy and performance, since loosening causes the data loss.

Cha et al. (2011a and 2011b) proposed a parallelization method to benefit from the computational power of multicore systems. Their method creates threads for S-functions that are specified to be parallelized by user. After the code generation step by Real-Time Workshop (RTW) tool of Simulink, some communication code among threads are inserted into the generated code. In another study of those authors, an optimization scheme of parallelizing Simulink blocks for building multi-threaded real-time applications on multicore systems was proposed (Cha, Kim and Kim 2012). That study extracts

dependency graph of a given block diagram and estimates execution time of Simulink blocks on target platforms. After that, some subtasks are created, a parallel C code is generated and the compiled program is allocated in the target multi-core system.

A model based parallelization method for Simulink models was proposed by Kumura et al. (2012). Their method provides to generate parallel C code from a model that is developed in Simulink and sequential C code that is generated by Simulink RTW tool. In that study, parallelization methods are classified into two: parallelization of one-step processing and multiple-step processing. Parallelization of one-step processing applies parallelization operations in an iteration of the main simulation loop; while parallelization of multi-step processing uses pipelining and such kind of techniques. Their study makes both kind of parallelization based on communicating sequential processes (CSP). Differently from (Canedo et al. 2010b), the original semantics of the model is preserved.

Umeda et al. (2016) proposed an automatic multigrain parallelization scheme for embedded applications by using OSCAR compiler. That study tackles both inter-block and intra-block parallelism by analyzing auto-generated C code, while earlier researches tackle intra-block parallelism (Umeda et al. 2016).

In addition to above studies, there are also some studies that consider mapping tasks to the CPU cores efficiently. Tuncali, Fainekos and Lee (2015, 2016) presented an approach that parallelize Simulink block diagrams of single and multi-rate control models. The method they proposed firstly extracts the dependency graph of a block diagram. Next, blocks are mapped to the CPU cores considering the dependency graph and the worst-case execution times of blocks. Finally, code generation and compilation is done for the target multi-core architecture.

Since Simulink is the most well-known model based development tool in the community, most of the studies are directed towards it. However, open source Scilab and its model-based development module, namely Xcos, look very promising for engineering simulation. Similar to Simulink, Xcos does not have a module for running simulations parallel, neither. Mukbil et al. (2016) published a study that presents an implementation strategy for parallel and distributed computing to Scilab and Xcos. That study is a first attempt to develop a distributed simulation toolbox for Scilab and Xcos.

From the viewpoint of Kumura et al. (2012), our parallelization approach targets one-step processing. Similar to Ozard and Desira (2000), we target to handle natural parallelism in simulation models. Our study differs from the others by (1) employing model transformation techniques for parallelization; (2) parallelizing simulations by handling only block diagram without modifying generated code or going to the compilation level; (3) targeting Xcos that is an open source and promising model-driven development tool for engineering simulations; (4) targeting all kind of simulations, not only embedded systems, that have been developed in Xcos. As far as we know, the number of current parallelization studies are very limited and most of them target Simulink. In this respect, we provide a generalized parallelization technique to parallelize models that have been developed via block diagrams. Besides, a Java based parallelization library for Scilab and blockset for Xcos are presented in our study to make parallel models be able to communicate with each other.

3 PARALLELIZATION WITH MODEL TRANSFORMATION

Model transformation is generally used for describing an existing model in another platform. In this study, we employ model transformation to parallelize existing simulations. In a few words, our approach requires developers to specify superblocks that should be parallelized. Then, the contents of the specified superblocks are replaced with appropriate communication blocks, while the original contents of those superblocks are carried into a new Xcos model. Lastly, the barrier blocks are also located for synchronization of the parallelized parts of the model. In the rest of this paper, we will refer to these

parallelized models as “parallel sub-model”. A parallel sub-model can be thought as the equivalent of a federate in High Level Architecture (HLA) or a logical process in parallel computing terminology.

3.1 Model Transformation Process

Our approach has three main steps, as shown in Figure 1. The details of each step are explained in following subsections.

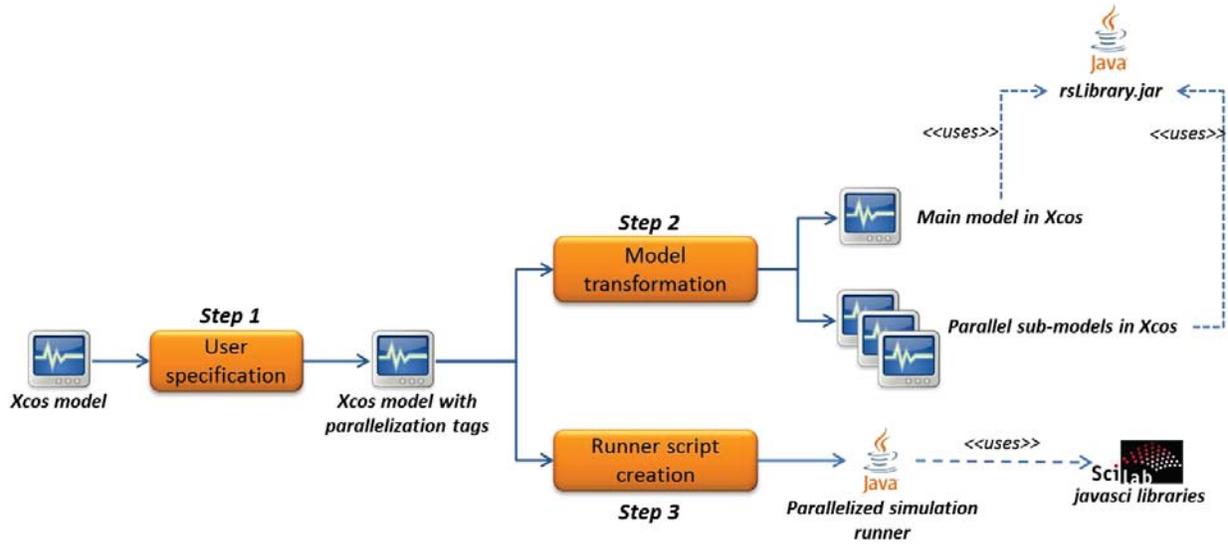


Figure 1: Parallelization process.

3.1.1 Pre-process: Parallelization Specification

Our approach is not fully automatic, instead it is user-assisted and requests developers to declare what superblocks/subsystems in the simulation will be parallelized. Therefore, the developer should provide some information about the component that will be parallelized. This information includes name of the new parallel sub-model that will be created, host name and port number that new parallel sub-model communicates with the main model. Parallelization specification can be performed by inserting some special tags to the superblocks in Xcos editor. In addition, the developer should specify host names and port numbers of all computing nodes that will join the simulation by using Xcos’s context setting feature. Although the parallelization specification requires some expert knowledge on this area, it can be done by many simulation developers by identifying natural parallelism. For instance, many developers can recognize that two different vehicles in a simulation can be simulated in parallel.

3.1.2 Model Transformation

The model that is saved in XML format by Scilab is handled by our model transformation rules and separated into parallel sub-models. The model transformation is completely independent from the number and type of input and output ports of the superblock that will be parallelized. Since Scilab does not have a communication facility, we have also implemented a library, namely rsLibrary, with Java socket programming facility and integrated it into Scilab. The rsLibrary provides developers to synchronously and asynchronously send/receive messages by using one of the User Datagram Protocol (UDP) and Transmission Communication Protocol (TCP). Besides, we developed communication blocks that connect Xcos and rsLibrary’s networking functions for both UDP and TCP. We preferred to employ TCP blocks in the parallelization phase, because it provides a reliable communication. The Xcos blocks that we defined and used in this study are as follows:

tcpSend block. *It sends a TCP message to the specified host and port number. Its input is the data that will be sent; while its output is a dummy value.*

tcpReceive block. *It receives a TCP message from the specified port number. This block is a preventing block that does not allow simulation engine to advance before receiving operation is completed. It has no input; while its output is received data.*

tcpIReceive block. *It is an asynchronous receiving operation and works just like Message Passing Interface's (MPI) ireceive function. It also receives a TCP message from the specified port number, but it does not prevent simulation engine to advance. Instead, it initiates the communication operation in a new thread. The output of this block is an integer that refers to the ID number of the asynchronous receiving operation. This ID generally goes to the barrier block as an input. With this ID, the barrier block can find the receiving operation that is waited.*

barrier block. *It is used after a tcpIReceive block. It is similar to MPI's barrier function. A barrier block prevents simulation engine to advance unless all input threads join. Input and output ports of a barrier block have the same sizes. Input ports are directed to the output ports unless they come from a tcpIReceive block. If an input comes from a tcpIReceive block, then barrier block waits until receiving operation is completed. Next, the received data is directed to the output port.*

submitJob block. *A submitJob block is the sequence of a tcpIReceive and a tcpSend block. It starts with an ireceive operation that receives the output of a parallel sub-model. Then, a dummy message is sent to that parallel sub-model to awake. The output of this block is the same with the output of tcpIReceive block.*

At the end of the model-transformation step, parallel sub-models are created with communication blocks that use rsLibrary. We preferred XML Stylesheet Transformation (XSLT) for model transformation just because of easy implementation. Instead of XSLT, more advanced transformation languages, such as Atlas Transformation Language (ATL), could be also preferred, but then the metamodel of Xcos has to be extracted.

Model transformation rules in this section are implemented for Xcos, but designed considering common properties of block diagrams in Simulink and SysML, as well. So, they can be easily adapted to other model-based languages that support block diagrams. Transformation for each parallel sub-model is done in two phases. The first phase is replacing communication blocks with the contents of superblocks that have been marked to be parallelized. The second phase is creating a new parallel sub-model that looks independent from the main simulation model. The basic rules that are executed during the first phase of transformation are here:

Rule 1. *“XcosDiagram_2_XcosDiagram” rule is the entry point of the model transformation. Elements under the root XML element in the source model are handled and redirected to other rules to be processed. Context information of the model is processed in this step, as well.*

Rule 2. *“Parallelization” rule works if a superblock that has been specified to be parallelized by developer is encountered. The responsibility of this rule is emptying the contents of the specified superblock and calling auxiliary rules to reconstitute it.*

Rule 3. *“CreateMuxBlock” rule is also called by “Parallelization” rule to pack input data of the parallelized superblock. The packed data is passed to the TcpSend block as a parameter.*

Rule 4. *“SuperBlock_2_CommunicationBlock” rule is called by “Parallelization” rule to locate a submitJob block into the parallelized superblock. This block provides coordination and synchronization between parallel sub-models and main model. An asynchronous receiving operation has been preferred*

in this block not to prevent parallelism. If a synchronous operation had been used, then simulation engine would be stucked in the receiving operation and not allow to the other parallel sub-models' execution.

Rule 5. *“CreateBarrier” rule inserts a Barrier block just before the parallelized superblock joins with other blocks. It prevents simulation engine to execute next blocks before the execution of parallel sub-models have been exactly completed and outputs are gathered.*

Rule 6. *“CreateExplicitLinks” rule is called by “Parallelization” rule, too. It re-organizes the connection links in the parallelized superblock.*

Rule 7. *“CreatePorts” is an auxiliary rule that creates and organizes input and output ports of newly created blocks. It is called by several rules that tries to create blocks with input or output ports.*

The second step is responsible for constituting the new parallel sub-model that communicates with the main model. In this step, we preferred using synchronous message receiving blocks, because the parallel sub-model will be enabled after it receives a signal from the main model. In addition to some of the rules above, following rules are also applied during the second step:

Rule 1. *“CreateParallelSubModel” rule is called when a superblock is handled, that has been specified to be parallelized by developer. It creates a new Xcos model diagram and calls other rules to constitute new parallel sub-model. Context information of the main model is also transferred into the parallel sub-model in this step.*

Rule 2. *“CreateCommunicationBlocks” rule is called by “CreateParallelSubModel” rule to locate communication blocks into the parallel sub-model. These communication blocks provide coordination and synchronization between parallel sub-models and main model.*

Rule 3. *“CreateDemuxBlock” rule is also called by “CreateParallelSubModel” rule to unpack received data from the main model.*

3.1.3 Post-process: Runner Creation

After the model-transformation step, there can be tens of parallel sub-models around. Manually initializing all of the parallel sub-models is a time-wasting operation. Moreover, developers have to be careful about the execution order. If the parallel sub-models are not initialized in the correct order, some of the messages may be lost. To this end, a runner script and a Java program are automatically created after the model-transformation, that are responsible for initialization of the parallel sub-models in the correct order. This Java program uses Scilab's application programmer's interface (API) for Java, namely javasci, to connect to a Scilab session. The developer should run that Java program, instead of starting Xcos simulation. Although Scilab does not allow to execute multiple Xcos simulations concurrently, created Java program and Scilab script overcome this problem by running multiple Scilab console application that is the same with Scilab without an advanced graphical user interface (GUI).

4 CASE STUDY

We experimented our parallelization method in an air defense simulation that contains two missiles as in Figure 2. This model is used as the source of model transformation process. The developer specifies a superblock that will become a parallel sub-model. In this study, Missile1 is selected to be parallelized. The specification is done by using some special characters and keywords in the definition of the superblock, so that model transformation process can handle the superblocks that will be parallelized. Name, IP address/hostname, and port number of the new parallel sub-model are specified as in Figure 2. Besides, the developer has to define a special variable, namely “mainPort”, to specify port number of the

main process by using Xcos's "Set Context" menu. The details of Missile1 superblock is given in Figure 3.

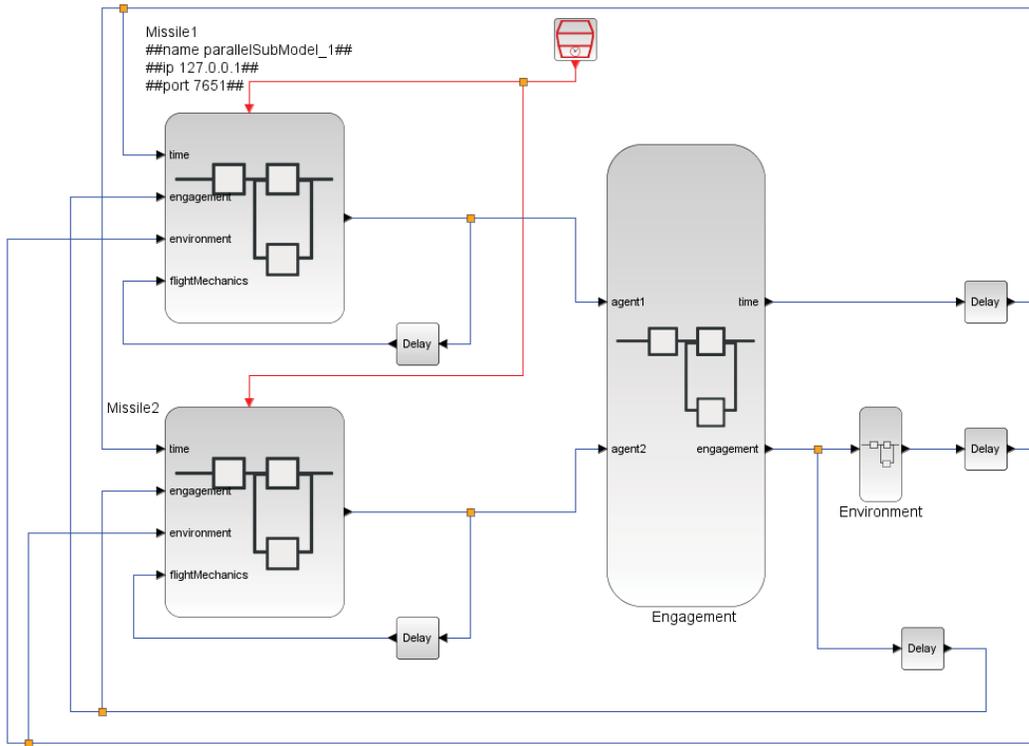


Figure 2: A sample model to be parallelized.

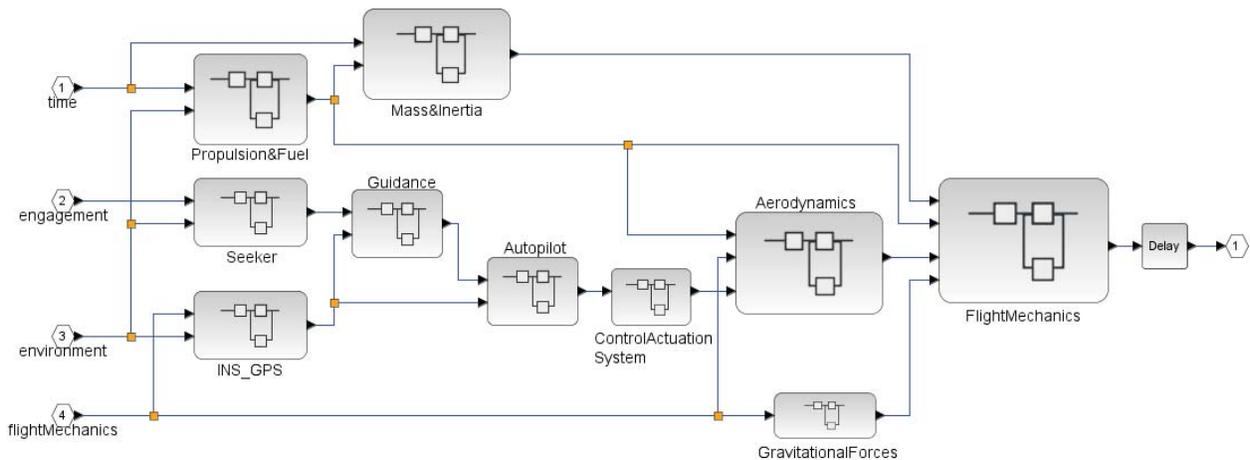


Figure 3: Inner structure of Missile1 superblock before model transformation.

After the model transformation process runs, there will be two separate models: main model (Figure 4 and 5) and parallel sub-model (Figure 6). Model transformation applies these modifications and creations respectively:

- Firstly, Missile1 superblock is emptied in the main model. Then, a multiplexer block and a submitJob block are inserted (Figure 5). The submitJob block firstly performs an asynchronous receiving operation. This asynchronous operation pretends like a message is received. Therefore, simulation engine can continue instead of waiting until the receiving operation is completed.

Next, the submitJob block sends a message to the parallel sub-model. The parallel sub-model is enabled when that message is received.

- In the main model, a barrier block is inserted just before the Engagement block where Missile1 and Missile2 joins (Figure 4). The barrier block forces simulation engine to wait until asynchronous receiving operations are completed. In this respect, receiving and barrier operations in our model run like MPI's ireceive and barrier operations.
- Finally, a new parallel sub-model is created to simulate Missile1, as in Figure 6. This model is executed separately from the main model, but it has to be enabled by receiving a message from the main model. Therefore, the model transformation inserts a synchronous receiving operation at the beginning of this model. When an iteration of this parallel sub-model is done, the output is sent to the main model.

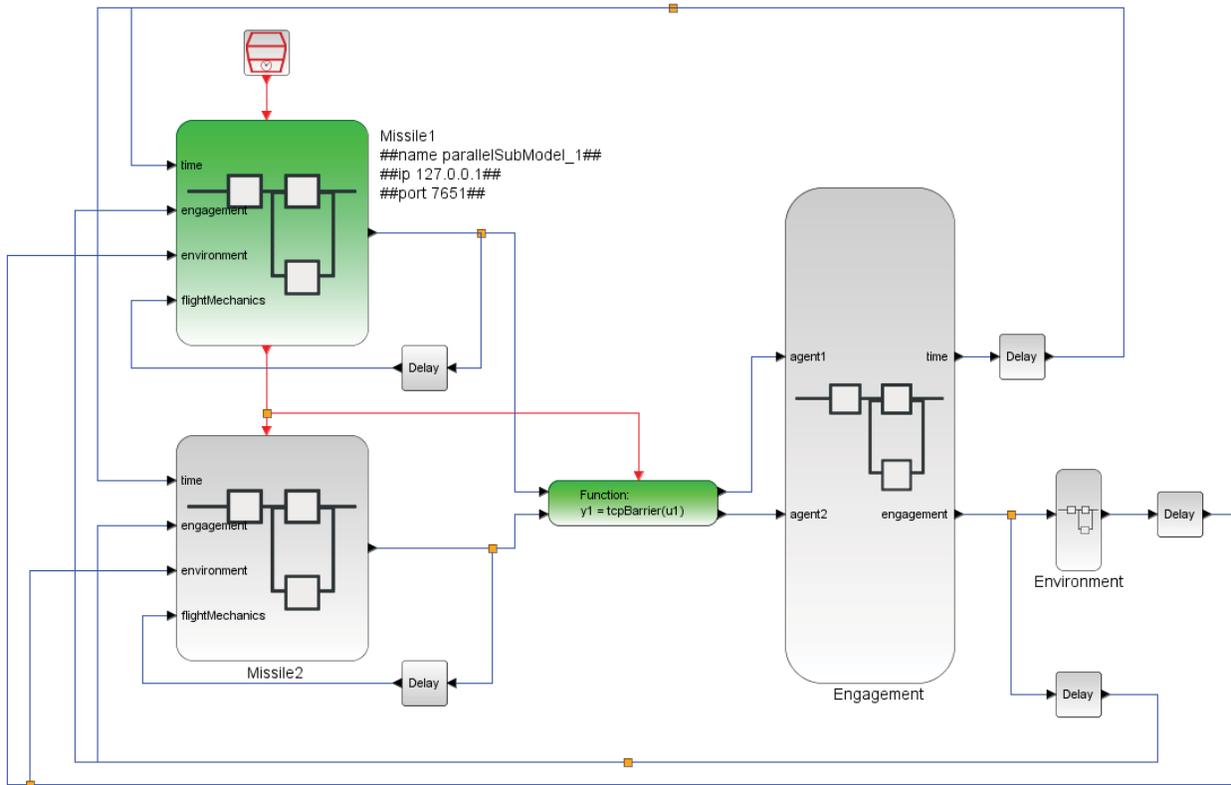


Figure 4: Main model after model transformation (the new and modified blocks are green colored).

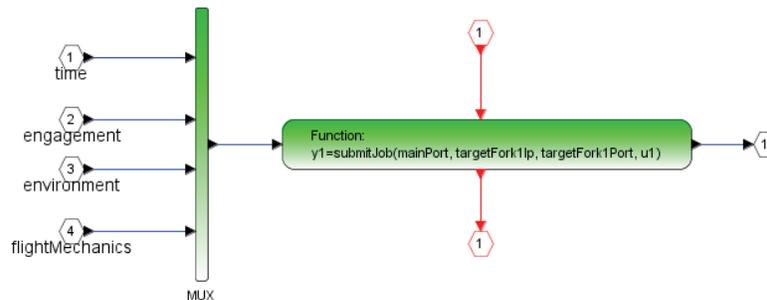


Figure 5: Inner structure of Missile1 superblock after model transformation (the new and modified blocks are green colored).

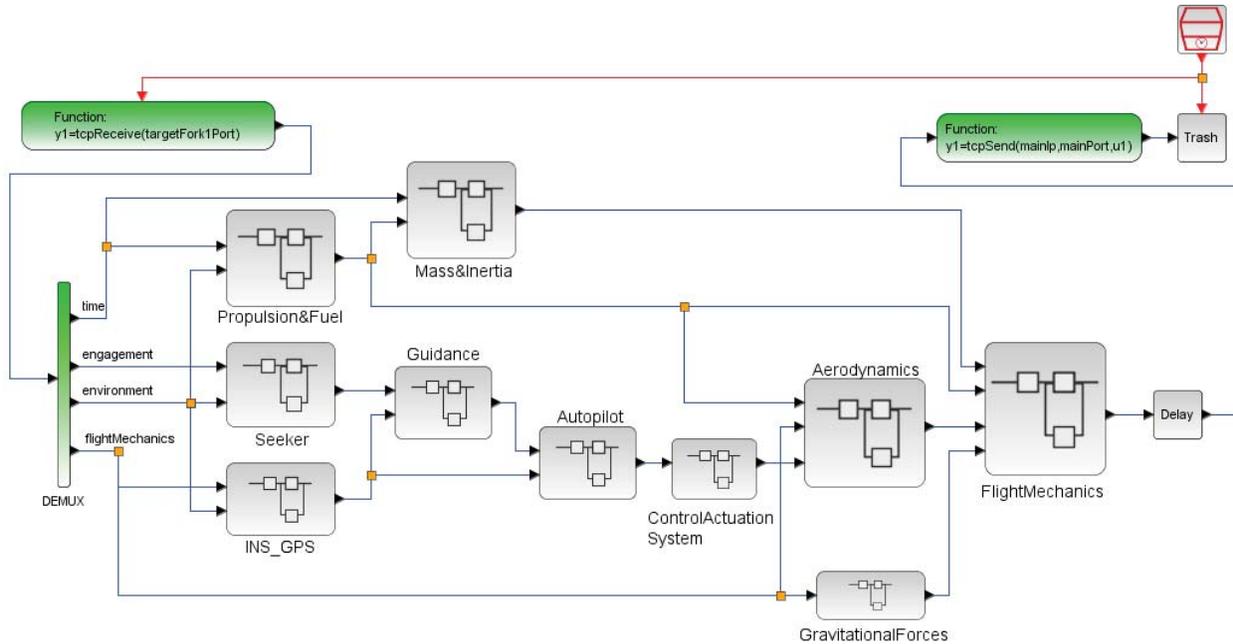


Figure 6: Created parallel sub-model for Missile1 (the new and modified blocks are green colored).

Asynchronous receiving operations and connections between command ports (red colored connections in Figure 4, 5 and 6) are key points of this study, because they force simulation engine to comply with the execution order that is determined by the model transformation rules. In this respect, Scilab engine is not aware the model runs in parallel. Regarding to the execution of the created parallel simulation system, the parallel sub-model is run firstly. Then, the main model is started and it enables the parallel sub-model by submitting a job. The global time of simulation is advanced by main model and parallel sub-models together.

After the model transformation, we observed the speedup of the created parallel simulation in a simple setup. We experimented the original model in a single core of a CPU with the frequency of 2.80 GHz. Then, we experimented the parallelized model in two cores of the same CPU. We simulated our model with different number of simulation time steps. In order to examine the behavior of our model under various computational loads, we increased or decreased the number of computations in some blocks, that can be also thought as changing simulation fidelity. Table 1 is a summary of our experiments that shows how long the models are simulated via serial and parallel execution; standard deviations of the execution times; and average speedups that are achieved by parallelization. Every experiment in Table 1 are performed at least 5 times. In our case study, only Missile1 superblock was selected to be parallelized, but more blocks can be parallelized. For instance, Propulsion & Fuel, Seeker and INS_GPS blocks under Missile1 in Figure 3 could be run in parallel. To keep the case study simple, we demonstrated the parallelization of only one superblock.

5 DISCUSSION

Simulation parallelization comes with several advantages. First of all, computation time can be reduced considerably and simulations that require too much resources can be executed in acceptable time periods. Correspondingly, simulations can be run much more times to obtain more accurate results, in case Monte Carlo simulations are required. To this end, we presented a simple approach to parallelize existing simulations that have been designed with block diagrams by using model transformation techniques. We tried to show block diagrams can be parallelized with minimum amount of knowledge. That is why we kept several things simple. For instance the communication among parallel sub-models are performed by

sending/receiving messages with socket programming API of Java, instead of more dedicated libraries such as MPI.

The method that we implemented provides a simple parallelization solution to developers. Thanks to it, researchers from various kind of disciplines can easily parallelize their simulation. The only thing that a developer has to do is determining time consuming components of the model and marking them as parallel sub-models. If parallelization specification is not done carefully, bad parallelization strategies can be shown up and the efficient usage of parallel resources cannot be benefited. However, rest of the parallelization process is fully-automatic and saves high amount of time and effort of developers. We observed that the generated parallel simulation can provide a speedup of 1.68 with 2 cores in our case study. According to the Table 1, the speedup increases to some extent while total time step or computational load increases, because communication cost is relatively high in shorter or lighter simulations comparing to the computational cost. However, after some The speedup value stays almost the same . The speedup can be also improved with a reasonable parallelization strategy, unless Amdahl's law (Amdahl 1967) is observed. On the other hand, the speedup value can be worse in much complex and larger models where communication could bottleneck the parallelization; but this is a well-known and inevitable problem of parallel processing (Amdahl 1967). Although our first goal is proving the concept of using model transformation for parallelization, the speedup is still very promising. In terms of memory usage, the proposed approach comes with ignorable overhead, since the only overhead is the memory consumption of primitive communication blocks that are put into the original model.

Table 1: Comparison of parallel and serial execution of the simulation.

Total time step	Computational Load	Parallel Execution Time and Standard Deviation	Serial Execution Time and Standard Deviation	Average Speedup
10	Low	5.70 seconds, $\sigma = 0.32$	7.69 seconds, $\sigma = 0.50$	1.35
50	Low	21.46 seconds, $\sigma = 1.01$	32.88 seconds, $\sigma = 0.81$	1.53
100	Low	42.23 seconds, $\sigma = 1.07$	64.34 seconds, $\sigma = 0.77$	1.52
10	Medium	10.54 seconds, $\sigma = 0.26$	13.94 seconds, $\sigma = 0.17$	1.32
50	Medium	39.50 seconds, $\sigma = 0.79$	66.43 seconds, $\sigma = 0.84$	1.68
100	Medium	81.36 seconds, $\sigma = 2.03$	132.86 seconds, $\sigma = 2.14$	1.63
10	High	18.70 seconds, $\sigma = 0.91$	28.36 seconds, $\sigma = 0.93$	1.52
50	High	81.62 seconds, $\sigma = 1.39$	130.08 seconds, $\sigma = 2.08$	1.59
100	High	154.79 seconds, $\sigma = 1.00$	254.95 seconds, $\sigma = 6.53$	1.65

At the end of model transformation, we validated our approach by comparing the results of generated model and original model. Both models produced exactly same results as expected, because the proposed approach transforms the input model without harming semantics of the simulation. In addition, synchronization between parallel sub-models guarantees to keep the original execution order. However, there are some problems at visualizing generated model. Although it can be executed without any problems, some of the generated blocks cannot be displayed neatly. The similar problem is also existing in some other model transformation studies, such as Ledett et al. (2015) and we think that it can be ignored as long as the correctness of the generated model is preserved.

In addition to Xcos, our proposed method can be applied to models that have been developed by using block diagrams, such as Simulink. The motivation that we chose Xcos is that it is a promising open-

source software. During the model transformation we preferred XSLT, just because of easy implementation. ATL or other advanced transformation languages could be also preferred, but then metamodels of Simulink and Xcos block diagrams had to be obtained. At the end of the model transformation, parallel sub-models that communicate with each other appear. These parallel sub-models can be also executed in different computers that are geographically distributed, because they use message-passing techniques to communicate. In this case, parallel sub-models should be deployed to distributed computation nodes manually.

6 FUTURE WORK

Our parallelization process is semi-automatic for now and dependent on the developer's specification. We would like to reduce developer's intervention to the original model for parallelization. Our plan is gaining the ability to detect the most time and resource consuming parts of the model and suggest a parallelization process to the user. Similar to the study of Cha, Kim and Kim (2012), detecting those parts can be performed by assigning predetermined numbers to each block types and finding subgraphs that have large values. It can be also done by monitoring sample runs and identifying where the bottlenecks are.

As a next step, we are planning to change communication module from basic socket programming to MPI that is one of the most well-known messaging standards in parallel processing community. Therefore, parallel sub-models will be able to use more advanced MPI functions, such as broadcasting, gathering scattering, reducing, etc.

In the long term, we are going to extend our approach with advanced time management mechanisms for PADS. Our plan is removing "Barrier" blocks and increase the usage of parallelism by employing the Time Warp algorithm (Jefferson 1985) that is the most well-known optimistic time management algorithm. Therefore, parallel sub-models will be allowed to make some mistakes in computation due to the late arriving messages from other parallel sub-models. When the message is lately received, the parallel sub-model is responsible for fixing the erroneous operation.

REFERENCES

- Amdahl, G. M. 1967. "Validity of the single processor approach to achieving large scale computing capabilities". In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, pp. 483–485.
- Canedo, A., T. Yoshizawa, and H. Komatsu. 2010a. "Automatic Parallelization of Simulink Applications". In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, pp. 151–159. Toronto, Ontario, Canada, ACM.
- Canedo, A., T. Yoshizawa, and H. Komatsu. 2010b. "Skewed Pipelining for Parallel Simulink Simulations". In *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 891–896, Dresden, Germany, European Design and Automation Association.
- Cha, M., K. H. Kim. 2011a. "Automatic Building of Real-Time Multicore Systems Based on Simulink Applications". In *Ubiquitous Computing and Multimedia Applications: Second International Conference, UCMA 2011, Daejeon, Korea, April 13-15, 2011. Proceedings, Part II*, edited by T. Kim, H. Adeli, R. J. Robles and M. Balitanas, pp. 209–220. Berlin, Springer Berlin Heidelberg.
- Cha, M., K. H. Kim, C. J. Lee, D. Ha, and B. S. Kim. 2011b. "Deriving High-Performance Real-Time Multicore Systems Based on Simulink Applications". In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pp. 267–274.
- Cha, M., S. K. Kim, and K. H. Kim. 2012. "An Automatic Parallelization Scheme for Simulink-based Real-Time Multicore Systems". In *Proceedings of Software Technology (SoftTech 2012)*, pp. 215–217, Cebu, Philippines, Advanced Science and Technology Letters.

- Fujimoto, R. M. 2016. “Research Challenges in Parallel and Distributed Simulation”. *ACM Transactions on Modeling and Computer Simulation* vol. 26, pp. 22:1–22:29.
- Hutchinson, J., J. Whittle, and M. Rouncefield. 2015. “Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure”. *Science of Computer Programming* vol. 89, pp. 144–161.
- Jefferson, D. R. 1985. “Virtual Time”. *ACM Transactions on Programming Languages and Systems* vol. 7, pp.404–425.
- Ledett, J., S. Çam, B. K. Görür, O. Dayıbaş, H. Oğuztüzün, and L. Yılmaz. 2015. “A Hybrid Transformation Process for Simulation Modernization and Reuse via Model Replicability and Scenario Reproducibility”. In *Proceedings of the 2015 Alabama International Simulation Conference*, Huntsville, Alabama, USA.
- Kalos, M. H., and P. A. Whitlock. 2008. *Monte Carlo Methods*. Weinheim, WILEY-VCH Verlag GmbH & Co. KGaA.
- Kumura, T., Y. Nakamura, N. Ishiura, Y. Takeuchi, and M. Imai. 2012. “Model based parallelization from the simulink models and their sequential C code”. In *Proceedings of the 17th Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI 2012)*, pp. 186–191.
- Mukbil, A., P. Stroganov, U. Durak, and S. Hartmann. 2016. “Towards a Distributed Simulation Toolbox for Scilab”. *Workshop der ASIM/GI Fachgruppen STS und GMMS*.
- Ozard, J., and H. Desira. 2000. “Simulink model implementation on multi-processors under Windows-NT”. *Simulation Series* vol. 32, pp. 197–204.
- Xcos. “Scilab/Xcos”. <http://www.scilab.org/scilab/features/xcos>. Accessed Dec. 02, 2016.
- Tuncali, C. E., G. Fainekos, and Y-H. Lee. 2015. “Automatic Parallelization of Simulink Models for Multi-core Architectures”. In *Proceedings of the 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conf on Embedded Software and Systems*, pp. 964–971, Washington, IEEE Computer Society.
- Tuncali, C. E., G. Fainekos, and Y-H. Lee. 2016. “Automatic Parallelization of Multirate Block Diagrams of Control Systems on Multicore Platforms”. *ACM Transactions on Embedded Computing Systems* vol 16, pp. 15:1–15:26.
- Umeda, D., T. Suzuki, H. Mikami, K. Kimura, and H. Kasahara. 2016. “Multigrain Parallelization for Model-Based Design Applications Using the OSCAR Compiler”. In *Revised Selected Papers of the 28th International Workshop on Languages and Compilers for Parallel Computing - Volume 9519*, pp. 125–139. Raleigh, NC, USA, Springer-Verlag New York, Inc.

AUTHOR BIOGRAPHIES

B. KAAAN GÖRÜR is a specialist computer engineer at Roketsan A.S. in Ankara, Turkey. He is also a PhD student in Department of Computer Engineering in Hacettepe University. His research interests include parallel and distributed simulation, model driven engineering, agent based modeling and simulation, and simulation visualization. His email address is kaan.gorur@roketan.com.tr.

A. NEBİ ÇALLI is an executive aerospace engineer at Roketsan A.S. in Ankara, Turkey. He is also an M.Sc. student in Department of Aerospace Engineering in Middle East Technical University. His research interests include modeling and simulation, system design, and flight sciences. His email address is nebi.calli@roketan.com.tr.