Automatic parallelization of multi-rate fmi-based co-simulation on multi-core

Salah Eddine Saidi, Nicolas Pernet, Yves Sorel

AUTOMATIC PARALLELIZATION OF MULTI-RATE FMI-BASED CO-SIMULATION ON MULTI-CORE

Salah Eddine Saidi
Nicolas Pernet

IFP Energies nouvelles
1 et 4 avenue de Bois Préau
Rueil-Malmaison, France
{salah-eddine.saidi, nicolas.pernet}@ifpen.fr

Yves Sorel

INRIA
2 rue Simone Iff
Paris, France
yves.sorel@inria.fr

ABSTRACT

Co-simulation refers to simulating a complex system using several coupled numerical models. Engineers define the rate of data exchange between the models by setting communication steps. FMI is a standardized interface which easily allows coupling and co-simulation of numerical models. The RCOSIM approach allows the parallelization on multi-core processors of co-simulations using the FMI standard. In this paper, we tackle the limitations of this approach. First, we extend the co-simulation to multi-rate, i.e. with different communication steps. We present graph transformation rules and an algorithm that allow the execution of each model at its respective rate while ensuring correct data exchange between models. Second, we present an acyclic orientation heuristic for handling mutual exclusion constraints between operations that belong to the same model due to the non-thread-safe implementation of FMI. We evaluate the obtained speedup on a multi-core processor and the effect on the accuracy of the numerical results.

Keywords: parallel, co-simulation, multi-core, acyclic orientation, scheduling.

1 INTRODUCTION

The recent advancement in merging different technologies and engineering domains has led to the emergence of the field of Cyber-Physical Systems (CPS). In such systems, embedded computers interact with, and control physical processes. Because of the heterogeneity of the involved components (multi-physics, sensors, actuators, embedded computers), CPS may feature very complex architectures and designs, ever raising the need for time, cost, and effort-effective approaches for building robust and reliable systems. Numerical simulation has proven successful in responding to this need, and is therefore increasingly considered to be an indisputable step in design verification and validation. For complex CPS, experts of different engineering disciplines may be involved in the design process by developing models for different parts of the system. In a later stage, the developed models need to be integrated into one simulation environment and coupled together in order to perform simulation at the system level. This approach is called co-simulation. Each model is assigned an integration step, which in some cases is driven by the dynamics of the modeled system and the control objective, and exchanges data with the other models according to a communication step which can be equal or larger than its integration step. Enabling co-simulation of heterogeneous models requires using adequate tools and methods. In this scope, the Functional Mock-up Interface (FMI) (Blochwitz et al. 2012) was developed with the goal of facilitating the co-simulation and the exchange of numerical models. FMI

SpringSim-TMS/DEVS, 2017 April 23-26, Virginia Beach, VA, USA;

defines a standardized interface that can be implemented by modeling tools in order to create models that can be connected with other FMI models. A model that is developed using FMI is exported as a Functional Mock-up Unit (FMU) which is a package that encapsulates an XML file containing, among other data, the definitions of the model's variables, and a library defining the equations of the model as C functions.

Co-simulation is an alternative approach to monolithic simulation where a complex system is modeled as a whole using differential equations and simulated by numerically solving these equations. In a CPS, the controlled physical process constitutes a multi-physics system and is modeled in the continuous-time domain using (hybrid) Ordinary Differential Equations (ODEs). Because they are aimed to be implemented on embedded computers, numerical laws that control the physical process are modeled in the discrete-time domain. All of these features add to the complexity of the numerical models. Co-simulation has a number of advantages over the monolithic approach. It allows modeling each part of the system using the most appropriate tool instead of using a single modeling software. Also, it allows a better intervention of the experts of different fields at the subsystem level. Furthermore, co-simulation facilitates the upgrade, the reuse, and the exchange of models. In co-simulation, the equations of each FMU are integrated using a solver separately. FMUs exchange data by updating their inputs and outputs according to their communication steps. It is worth noting that in this paper, the term co-simulation is used to refer to the simulation of FMUs generated from FMI for Model Exchange, FMI for Co-Simulation, or a combination of both types. In our approach, when importing Model Exchange FMUs, a dedicated solver is assigned for each FMU. For connected FMUs with different communication steps, the master algorithm can provide extrapolation techniques to produce the missing data. In the following, we consider FMI co-simulation without step rejection.

Figure 1 shows the evolution of time and data exchange between three FMUs. The vertical double arrows represent data exchange between the FMUs. FMU B must be assigned a communication step H_B equal to the smaller communication step among H_A and H_C since it exchanges data with both FMU A and FMU B. Co-simulation requires domain-specific information for each involved FMU. Such information, which is beyond the scope of FMI, can be provided by domain experts, for example by using an approach like the one propped by Sirin et al. (2015). The communication steps of FMUs can be shared as part of this information.

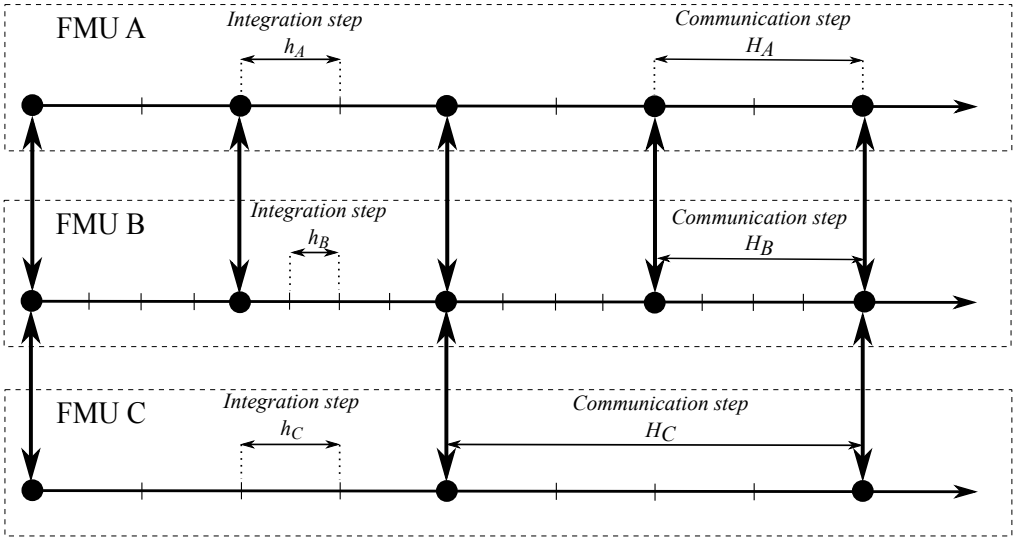


Figure 1: Evolution of time and data exchange between three FMUs during co-simulation

Usually, assembling FMUs results in a heavy co-simulation, requiring high processing power. Increasing CPU performance through frequency scaling has reached some technological limits. As a consequence,

during the last decade, parallelism has been by far the main way for increasing the performance of processors. In fact, the last years have witnessed a major shift among semiconductor manufacturers to building multi-core processors, i.e. integrating multiple processors into one chip allowing parallel processing on a single computer. Enabling parallel execution of heavy co-simulations on multi-core processors is keenly sought by the developers and the users of simulation tools. However, fulfilling this objective is not trivial and appropriate parallelization techniques need to be applied on co-simulation models in order to accelerate their execution on multi-core processors.

In order to achieve (co-)simulation acceleration using multi-core execution, different approaches are possible and were already explored. From a user point of view, it is possible to modify the model design in order to prepare its multi-core execution, for example by using models that are developed using parallel computing libraries as in (Gebremedhin et al. 2012). Using parallel solvers as in (Elmqvist et al. 2014) is another alternative. (Galtier et al. 2015) presented the DACCOSIM framework which allows multi-threaded distributed simulation of FMUs on multi-core processors as well as clusters of computers. However, the parallelization achieved through this approach is not automatic as the user has to define the distribution of the FMUs on the architecture. A more detailed discussion of related work is given in (Ben Khaled et al. 2014).

In this paper, we address the problem from a co-simulation tool provider point of view. In such a tool, the user connects different FMUs, embedding solvers or using solvers provided by the co-simulation tool. In this case, it is not possible to change the models, the solvers, nor the modeling tools. The Refined CO-SIMulation (RCOSIM) (Ben Khaled et al. 2014) approach allows the parallelization on multi-core processors of co-simulations using the FMI standard, by generating an efficient multi-core FMI master. This approach is based on a representation of the co-simulation using a Directed Acyclic Graph (DAG). Each vertex of this graph represents an operation which is an atomic unit of execution consisting of one or several FMU C function calls. Each edge represents a precedence relation between two operations. In this paper, we deal with the limitations of this approach that we highlighted in a recent work (Saidi et al. 2016). Although RCOSIM resulted in interesting co-simulation speedups, it has two limitations that have to be considered in the multi-core scheduling problem in order to obtain better performances. First, RCOSIM supports only mono-rate co-simulations, i.e. the different FMUs have to be assigned the same communication step. Indeed, because of this limitation, RCOSIM is unable to handle an important number of industrial applications containing multiple communication steps as provided by the engineers. Second, the functions of an FMU may not be thread-safe, i.e. they cannot be executed in parallel as they may share some resource (e.g. variables) that might be corrupted if two operations try to use it at the same time. Consequently, if two or more operations belonging to the same FMU are executed on different cores, a mechanism that ensures these operations are executed in disjoint time intervals must be set up. Previously, these mutual exclusion constraints were tackled in RCOSIM by executing the operations of the same FMU on the same core which restricts the exploitation of the parallelism.

We propose in this paper a solution for each of the aforementioned limitations by making extensions to RCOSIM. In order to allow handling multi-rate co-simulations, we present a graph transformation algorithm that is applied to the initial constructed graph. Then, mutual exclusion constraints are represented by adding, in the graph, non oriented edges between operations belonging to the same FMU. We present an acyclic orientation heuristic that is used to assign directions to the added non oriented edges. The proposed solutions are implemented in the xMOD multi-model integration software. We evaluate the obtained speedup on a multi-core processor and assess the effect on the accuracy of the numerical results.

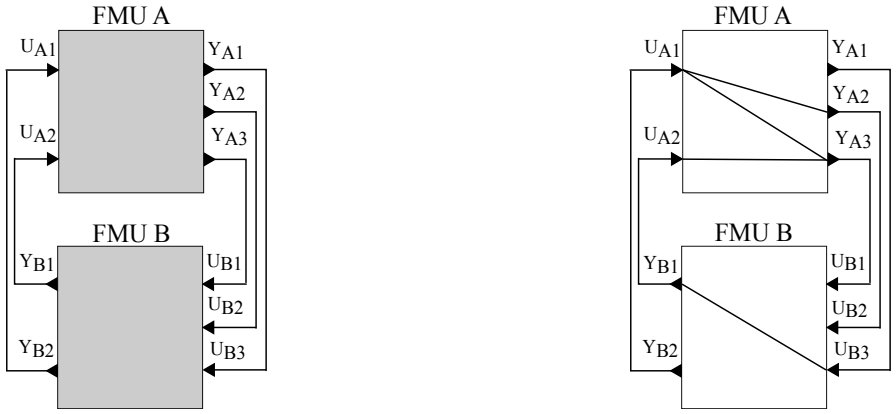
The rest of the paper is organized as follows. The next section gives an overview of the RCOSIM method and the proposed extensions. In section 3, graph transformation rules and an algorithm for handling multi-rate co-simulation are presented. In section 4, an acyclic orientation heuristic is presented. This heuristic is meant to handle the mutual exclusion constraints between the operations that belong to the same FMU. Section 5 describes the allocation and scheduling heuristic. In section 6, we present the obtained results.

We give a brief description of the industrial use case that is used to assess the proposed method and then present the obtained speedup and the numerical error evaluation. Finally, we conclude with some remarks and perspectives for future work in section 7.

2 OVERVIEW OF RCOSIM

The proposed method is an extension of the RCOSIM approach. We briefly present RCOSIM in this section before giving a description of the proposed extensions to tackle its limitations.

The entry point is a user-specified set of interconnected FMUs as depicted in Figure 2a. The execution of each FMU is seen as computing a set of input operations (one operation for each of the inputs of the FMU), a set of output operations (one operation for each of the outputs of the FMU), and one state operation for updating the state variables of the FMU. An operation is defined by a number of FMU C function calls. An input (resp. output) operation is executed by calling *fmiSet* (resp. *fmiGet*) function and a state operation is executed by calling *SetTime, GetDerivatives, SetContinuousStates, etc.*, functions in the case of FMI for Model Exchange or *DoStep* function in the case of FMI for Co-Simulation. Thanks to FMI, it is additionally possible to access information about the internal structure of a model encapsulated in an FMU. In particular, as shown in Figure 2b, FMI allows the identification of Direct Feedthrough (e.g. Y_{B1}) and Non Direct Feedthrough (e.g. Y_{A1}) outputs of an FMU and other information depending on the version of the standard. RCOSIM consists in using FMU information on input/output dependencies to build a graph with an increased granularity, and then exploiting the potential parallelism (Saidi et al. 2016) of this transformed graph by using a heuristic to build an off-line multi-core schedule.



(a) Inter-FMU dependencies specified by the user

(b) Intra-FMU dependencies provided by FMI

Figure 2: An example of inter and intra-FMU dependencies of two FMUs connected by the user

The co-simulation can be described by a DAG $G_I(V_I, A_I)$ called the operation graph where each vertex $o_i \in V_I$ represents one operation and each oriented edge $(o_i, o_j) \in A_I$, that we call an arc, represents a precedence relation between two operations. The operation graph is built by exploring the relations between the FMUs and between the operations of the same FMU. A vertex is created for each operation and arcs are then added between vertices if a data dependency exists between the corresponding operations. If FMI 1.0 is used, which does not give information about the dependencies between the state variables computation and the input and output variables computations, we must add edges between all input operations and the state operation of the same FMU. Furthermore, edges connect all output operations and the state operation of the same FMU because the computation at the simulation step k of an output must be performed with the same value of the state as for all the outputs belonging to the same FMU. Running the co-simulation consists in executing the graph repeatedly. A new execution of the graph cannot be started unless the previous one

was totally finished. The operation graph corresponding to the FMUs of Figure 2 is shown in Figure 3. In order to accelerate the co-simulation, the approach proceeds in two main phases: first, the operation graph is constructed by exploring the inter and intra-FMU dependencies and then, the operations are allocated to the available cores in such a way to minimize the makespan of the graph. The makespan corresponds to the execution time of the whole graph.

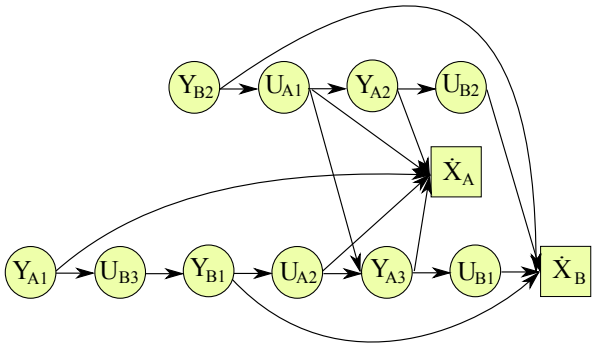


Figure 3: Operation graph obtained from the FMUs of Figure 2

We use the notation $M(o_i)$ to refer to the FMU to which the operation o_i belongs, and $T(o_i)$ to denote the type of the operation o_i, i.e. $T(o_i) \in \{input, output, state\}$. Once the operation graph is constructed, each operation o_i is characterized by a set of parameters. A profiling phase allows measuring the execution time $C(o_i)$. For each operation, the average execution time of multiple co-simulation runs is used. When real-time execution is aimed, Worst Case Execution Times (WCET) can be used instead. An operation o_i is characterized by its communication step $H(o_i)$ which is equal to the communication step of its FMU. The set of successors (resp. predecessors) of an operation o_i is denoted $\Gamma(o_i)$ (resp. $\overline{\Gamma}(o_i)$). In addition, the following parameters are defined for each operation o_i: the earliest start time from the graph beginning $S(o_i) = \max_{o_j \in \overline{\Gamma}(o_i)}(E(o_j))$ $(0 \; if \; \overline{\Gamma}(o_i) = \emptyset)$, the earliest end time from the graph beginning $E(o_i) = S(o_i) + C(o_i)$, the latest end time from the graph end $\overline{E}(o_i) = \max_{o_j \in \Gamma(o_i)}(\overline{S}(o_j))$ $(0 \; if \; \Gamma(o_i) = \emptyset)$, the latest start time from the graph end $\overline{S}(o_i) = \overline{E}(o_i) + C(o_i)$, and the flexibility $F(o_i) = R - S(o_i) - C(o_i) - \overline{E}(o_i)$. $R = \max_{o_i \in V_I}(E(o_i))$ denotes the critical path of the graph. These notations are used through the different phases of the proposed method in this paper.

Sections 3 and 4 describe new transformations that we propose to apply to the operation graph for handling multi-rate FMUs and mutual exclusion constraints between the operations beloning to the same FMU, respectively. Finally, the allocation of the different operations to the cores is achieved using a multi-core scheduling heuristic.

3 TRANSFORMATION ALGORITHM FOR MULTI-RATE GRAPHS

This section presents an algorithm that transforms the initial multi-rate operation graph $G_I(V_I, A_I)$ into a mono-rate operation graph $G_M(V_M, A_M)$. The aim of this transformation is to ensure that each operation is executed according to the communication step of its respective FMU and also to maintain a correct data exchange between the different FMUs, whether they have different or identical communication steps. Similar algorithms have been used in the real-time scheduling literature (Ramamritham 1995).

We define the *Hyper-Step (HS)* as the least common multiple (lcm) of the communication steps of all the operations: $HS = lcm(H(o_1), H(o_2), \ldots, H(o_n))$ where $n = |V_I|$ is the number of operations in the initial graph. The Hyper-Step is the smallest interval of time for describing a repeatable pattern of all the operations. The transformation algorithm consists first of all in repeating each operation o_i, r_i times where r_i is called the repetition factor of o_i and $r_i = \frac{HS}{H(o_i)}$. Each repetition of the operation o_i is called an occurrence

of o_i and corresponds to the execution of o_i at a certain simulation step. We use a superscript to denote the number of each occurrence, for instance o_i^s denotes the s^{th} occurrence of o_i. Then, arcs are added between operations following the rules presented hereafter. Consider two operations $o_i, o_j \in V_I$ connected by an arc $(o_i, o_j) \in A_I$, then adding an arc (o_i^s, o_j^u) to A_M, depends on the simulation steps (time) at which o_i^s and o_j^u are executed. In other words, if k_s and k_u are the simulation steps associated with o_i^s and o_j^u respectively, then the inequality $k_s \leq k_u$ is a necessary condition to add the arc (o_i^s, o_j^u) to A_M. In addition, o_j^u is connected with the latest occurrence of o_i that satisfies this condition, i.e. with the occurrence o_i^s such that $s = \max(0, 1, \ldots, r_i - 1) : k_s \leq k_u$. In the case where $H(o_i) = H(o_j)$ (and therefore $r_i = r_j$), occurrences o_i^s and o_j^u which correspond to the same number, i.e. $s = u$, are connected by an arc. On the other hand, if $H(o_i) \neq H(o_j)$, we distinguish between two types of dependencies: we call the arc $(o_i, o_j) \in A_I$ a *slow to fast* (resp. *fast to slow*) dependency if $H(o_i) > H(o_j)$ (resp. $H(o_i) < H(o_j)$). For a slow to fast dependency $(o_i, o_j) \in A_I$, one occurrence of o_i is executed while several occurrences of o_j are executed. In this case, arcs are added between each occurrence $o_i^s : s \in \{0, 1, \ldots, r_i - 1\}$, and the occurrence o_j^u such that:

$$u = \left\lceil s \times \frac{H(o_i)}{H(o_j)} \right\rceil .$$

We recall that for a slow to fast dependency, the master algorithm can preform extrapolation of the inputs of the receiving FMU. For a fast to slow dependency $(o_i, o_j) \in A_I$, arcs are added between each occurrence o_i^s, and the occurrence $o_j^u : u \in \{0, 1, \ldots, r_j - 1\}$ such that:

$$s = \left\lfloor u \times \frac{H(o_j)}{H(o_i)} \right\rfloor .$$

Arcs are added also between the occurrences of the same operation, i.e. $(o_i^s, o_i^{s'})$ where $s \in \{0, 1, \ldots, r_i - 2\}$ and $s' = s + 1$. Finally, for each FMU, arcs are added between the s^{th} occurrence of the state operation, where $s \in \{0, 1, \ldots, r_i - 2\}$, and the $(s+1)^{th}$ occurrences of the input and output operations. The multi-rate graph transformation is detailed in Algorithm 1. The algorithm traverses all the graph by applying the aforementioned rules in order to transform the graph and finally stops when all the nodes and the edges have been visited.

Figure 4 shows the graph obtained by applying the multi-rate transformation algorithm on the graph of Figure 3. In this example $H_B = 2 \times H_A$, where H_A and H_B are the communication steps of FMUs A and B respectively.

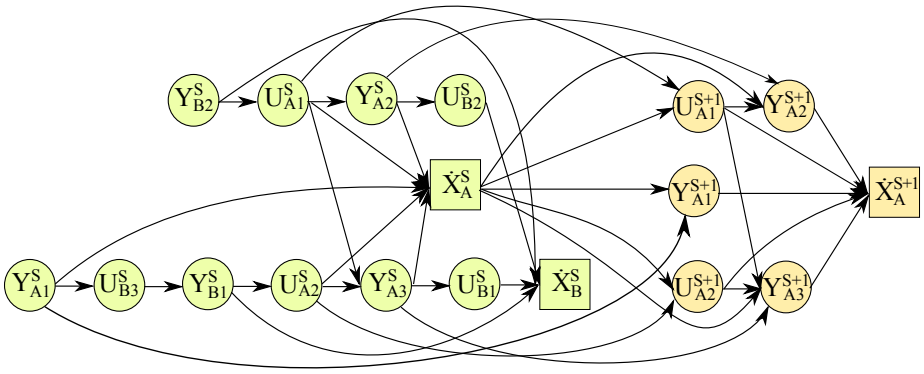


Figure 4: Graph obtained by applying the multi-rate transformation algorithm on the graph of Figure 3

Algorithm 1: Multi-rate graph transformation algorithm

Input: Initial operation graph $G_I(V_I, A_I)$;
Output: Transformed operation graph $G_M(V_M, A_M)$;
Set $G_I(V_I, A_I)$ the initial multi-rate operation graph and $G_M(V_M, A_M)$ the new mono-rate graph;
$V_M \leftarrow \emptyset$; $A_M \leftarrow \emptyset$;
foreach *operation* $o_i \in V_I$ **do**
  Compute the repetition factor of o_i: $r_i \leftarrow \frac{HS}{H(o_i)}$;
  Repeat the operation o_i: $V_M \leftarrow V_M \cup \{o_i^s\}, s \in \{0, \ldots, r_i - 1\}$;
end
foreach *arc* $(o_i, o_j) \in A_I$ **do**
  if $H(o_i) > H(o_j)$ *((o_i, o_j) is a slow to fast dependency)* **then**
    Compute $u = \left\lceil s \times \frac{H(o_i)}{H(o_j)} \right\rceil$ and add the arc (o_i^s, o_j^u) to the new graph $G_M(V_M, A_M)$:
    $A_M \leftarrow A_M \cup \{(o_i^s, o_j^u)\}, s \in \{0, \ldots, r_i - 1\}$;
  end
  else if $H(o_i) < H(o_j)$ *((o_i, o_j) is a fast to slow dependency)* **then**
    Compute $s = \left\lfloor u \times \frac{H(o_i)}{H(o_j)} \right\rfloor$ and add the arc (o_i^s, o_j^u) to the new graph $G_M(V_M, A_M)$:
    $A_M \leftarrow A_M \cup \{(o_i^s, o_j^u)\}, u \in \{0, \ldots, r_j - 1\}$;
  end
  else
    Add the arc (o_i^s, o_j^s) to the new graph $G_M(V_M, A_M)$: $A_M \leftarrow A_M \cup \{(o_i^s, o_j^s)\}, s \in \{0, \ldots, r_i - 1\}$;
  end
end
foreach *operation* $o_i \in V$ **do**
  Add an arc between successive occurrences of o_i: $A_M \leftarrow A_M \cup \{(o_i^s, o_i^{s+1})\}, s \in \{0, \ldots, r_i - 2\}$;
end
foreach *operation* $o_i \in V$ *such that* $T(o_i) = state$ **do**
  Add arcs (o_i^s, o_j^{s+1}) to the new graph $G_M(V_M, A_M)$ between o_i and every operation o_j such that
  $M(o_j) = M(o_i)$ and $T(o_j) \in \{input, output\}$: $A_M \leftarrow A_M \cup \{(o_i^s, o_j^{s+1})\}, s \in \{0, \ldots, r_i - 2\}$;
end

4 ACYCLIC ORIENTATION HEURISTIC FOR HANDLING MUTUAL EXCLUSION CONSTRAINTS

We describe in this section a heuristic for handling mutual exclusion constraints between operations belonging to the same FMU. These constraints are induced from the non thread-safe implementation of FMI. Without loss of generality, the superscript which denotes the number of the occurrence of an operation is not used in the rest of the paper for the sake of clarity. Each vertex of the graph $G_M(V_M, A_M)$ represents an operation that we refer to using the notation o_i.

In addition to the precedence relations represented by arcs, mutual exclusion relations are represented by (non oriented) edges in the mixed graph $G_X(V_X, A_X, D_X)$, where V_X is the set of vertices (operations), A_X the set of arcs, and D_X the set of edges. This mixed graph is obtained from the graph $G_M(V_M, A_M)$ by adding edges between operations belonging to the same FMU. Note that $V_X = V_M$. A precedence relation between the pair of operations o_i and o_j is denoted by an arc $(o_i, o_j) \in A$ which describes an order of execution between o_i and o_j, whereas a mutual exclusion relation between the pair of vertices o_p and o_q is denoted by

an edge $[o_p, o_q] \in D_X$ which means that o_p and o_q have to be executed in disjoint time intervals but in either order ($[o_p, o_q] = [o_q, o_p]$). Choosing one order or another does not impact the numerical results of the co-simulation since these operations do not have data dependencies. Still, we have to ensure mutual exclusion between them due to the non-thread-safe implementation of FMI. The goal of the heuristic proposed here is to assign directions to the edges of D_X in order to define an order of execution between each pair of operations joined by an edge with the aim of minimizing the length of the critical path of the resulting graph, and subject to the constraint that no cycle is generated in the obtained graph as the term acyclic orientation suggests. Indeed, the existence of a cycle in the graph prevents the definition of an execution order for the operations belonging to this cycle, and leads to a deadlock during the execution. The output of the heuristic is a graph $G_F(V_F, A_F, \emptyset) = G_F(V_F, A_F)$. The problem of acyclic orientation of mixed graphs has been studied and shown to be NP-Hard in (Andreev et al. 2000, Sotskov et al. 2002, Al-Anzi et al. 2006). In these studies, only the case of operations with unit (or equal) execution times were considered and the acyclic orientation problem was interpreted as a graph coloring problem. In this paper, we consider operations with different execution times and use an approach which takes into account that only operations belonging to the same FMU are joined by edges.

The orientation of the edges of D_X means applying a function $\alpha : \{[o_i, o_j] \in D_X\} \rightarrow \{(o_i, o_j), (o_j, o_i)\}$. Without loss of generality, consider that $\alpha([o_i, o_j]) = (o_i, o_j)$, then o_i is added to $\overline{\Gamma}(o_j)$ and o_j is added to $\Gamma(o_i)$. Consequently, the inequality $E(o_i) \leq S(o_j)$ holds and therefore o_i and o_j are executed in disjoint time intervals. The first step of the heuristic consists in building the mixed graph $G_X(V_X, A_X, D_X)$ by adding edges between operations belonging to the same FMU. The output of the heuristic is a topological ordering of the operations of each FMU. This can be interpreted for each FMU $\mathscr{M}$ as an assignment of integers $\phi : V_F \rightarrow \{1, 2, \ldots, m\}$ defining the order of execution for these operations, where m is the number of operations of the FMU $\mathscr{M}$. We call the integer assigned to an operation its rank.

Algorithm 2 details the acyclic orientation heuristic. After adding the edges to the graph, the heuristic iteratively assigns ranks to the operations. It keeps a list of already ordered operations l for each FMU $\mathscr{M}$ and at each main iteration it selects the operation which has the earliest start time $S(o_i)$ to be added to the list of its FMU. Ties are broken by selecting the operation with the least flexibility $F(o_i)$. We call the operation to be ordered at a given iteration, the pending operation. The the pending operation o_i is assigned the ranks $\tau \in \{x, x+1, \ldots, length(l)+1\}$ in ascending order, where $x = \max_{o_j \in \overline{\Gamma}(o_i)}(\phi(o_j)+1)$ (1 *if* $\overline{\Gamma}(o_i) = \emptyset$). For each rank τ, the assignment $\phi(o_i) = \tau$ is made and the ranks assigned to all the already ordered operations $o_{i'} \in l$ such that $\phi(o_{i'}) \geq \phi(o_i)$ are increased $\phi(o_{i'}) = \phi(o_{i'}) + 1$. Then, every edge $[o_i, o_i'] \in D_X : o_i' \in l$ is assigned a direction from the operation with the lower rank to the operation with the higher rank. At this point, the increase in R, the critical path of the graph, is evaluated. Next, the operations $o_{i'} \in l$ such that $\phi(o_{i'}) > \phi(o_i)$ are reassigned their previous ranks $\phi(o_{i'}) = \phi(o_{i'}) - 1$, and the pending operation is assigned the next rank $\phi(o_i) = \phi(o_i) + 1$. The increase in the critical path is evaluated again similarly. After repeating this for all the ranks $\tau \in \{x, x+1, \ldots, length(l)+1\}$, the pending operation is finally ordered at the rank which leads to the least increase in the critical path and edges $[o_i, o_i'] \in D_X : o_i' \in l$ are assigned directions as described before. The heuristic begins another iteration by selecting a new operation to be ordered. The heuristic assigns a rank to one operation at each iteration. Every operation is assigned a rank higher than the ranks of all its predecessors which guarantees that no cycle is generated. The heuristic finally stops when all the operations have been assigned ranks.

5 ALLOCATION AND SCHEDULING HEURISTIC

In order to achieve fast execution of the co-simulation on a multi-core processor, an efficient allocation and scheduling of the operation graph has to be performed. A multi-core scheduling heuristic is applied on the final graph $G_F(V_F, A_F)$ obtained after the acyclic orientation phase. Because of the number of fine-grained operations, and since the execution times and the dependencies between the operations are known before

Algorithm 2: Acyclic orientation heuristic

Input: Operation graph $G_M(V_M, A_M)$;
Output: New operation graph $G_F(V_F, A_F)$;
Set $G_M(V_M, A_M)$ the operation graph, $G_X(V_X, A_X, D_X)$ the mixed graph, and $G_F(V_F, A_F)$ the final graph;
$V_X \leftarrow V_M$; $A_X \leftarrow A_M$; $D_X \leftarrow \emptyset$; $V_F \leftarrow V_M$; $A_F \leftarrow A_M$;
Set Ω the set of all the operations: $\Omega \leftarrow V_X$;
foreach *pair* o_i, o_j *such that* $M(o_i) = M(o_j)$ **do** Add the edge $[o_i, o_j]$ to D_X: $D_X \leftarrow D_X \cup \{[o_i, o_j]\}$;
while $\Omega \neq \emptyset$ **do**
    Select the operation $o_i \in \Omega$ whose start time $S(o_i) = \max_{o_j \in \Omega}(S(o_j))$. Break ties by selecting the operation with the least flexibility;
    Set $\mathscr{M} \leftarrow M(o_i)$ and l the list of the already ordered operations of FMU $\mathscr{M}$;
    Set $\sigma \leftarrow \infty$; (Initialize the increase in the critical path);
    Set $x = \max_{o_j \in \overline{\Gamma}(o_i)}(\phi(o_j) + 1)$ $(1 \; if \; \overline{\Gamma}(o_i) = \emptyset)$;
    for $\tau = x$ *to* $length(l) + 1$ **do**
        $\phi(o_i) = \tau$;
        foreach $o_{i'} \in l$ *such that* $\phi(o_{i'}) \geq \phi(o_i)$ **do** $\phi(o_{i'}) \leftarrow \phi(o_{i'}) + 1$;
        foreach $o_{i'} \in l$ **do**
            if $\phi(o_{i'}) > \phi(o_i)$ **then** Assign a direction to the edge $[o_i, o_{i'}] \in D_X : \alpha([o_i, o_{i'}]) \leftarrow (o_i, o_{i'})$;
            else Assign a direction to the edge $[o_i, o_{i'}] \in D_X : \alpha([o_i, o_{i'}]) \leftarrow (o_{i'}, o_i)$;
        end
        Compute the new critical path and set σ' the increase in the critical path;
        if $\sigma' < \sigma$ **then** $rank \leftarrow \tau$, $\sigma \leftarrow \sigma'$;
        foreach $o_{i'} \in l$ *such that* $\phi(o_{i'}) > \phi(o_i)$ **do** Reassign $o_{i'}$ its previous rank: $\phi(o_{i'}) \leftarrow \phi(o_{i'}) - 1$;
    end
    $\phi(o_i) = rank$;
    foreach $o_{i'} \in l$ **do**
        if $\phi(o_{i'}) > \phi(o_i)$ **then** Assign a direction to the edge $[o_i, o_{i'}] \in D_X : \alpha([o_i, o_{i'}]) \leftarrow (o_i, o_{i'})$;
        else Assign a direction to the edge $[o_i, o_{i'}] \in D_X : \alpha([o_i, o_{i'}]) \leftarrow (o_{i'}, o_i)$;
    end
    Remove o_i from Ω;
end

runtime, it is more convenient to use an offline scheduling heuristic which has the advantage of introducing lower overhead than online scheduling heuristics. We use an offline scheduling heuristic similar to the one proposed in (Grandpierre et al. 1999) which is a fast greedy algorithm whose cost function corresponds well to our minimization objective. The heuristic considers the different timing parameters of each operation $o_i \in V_F$ in order to compute a schedule that minimizes the makespan of the graph. The heuristic schedules the operations of the graph $G_F(V_F, A_F)$ on the different cores iteratively and aims at minimizing a cost function which expresses the schedule pressure of an operation on a specific core. The schedule pressure is the difference between the makespan increase, by allocating this operation to this core, and the operation's flexibility. The heuristic updates the set of candidate operations to be scheduled at each iteration. An operation is added to the set of candidate operations if it has no predecessor or if all its predecessors have already been scheduled. For each candidate operation, the schedule pressure is computed on each core and the operation is allocated to its best core, the one that minimizes the pressure. Then, a list of candidate operation-best core pairs is obtained. Finally, the operation with the largest pressure on its best core is selected and scheduled. Synchronization operations are added between the scheduled operation and all its

predecessors that were allocated to different cores. The heuristic repeats this procedure and finally stops when all the operations have been scheduled.

6 EVALUATION

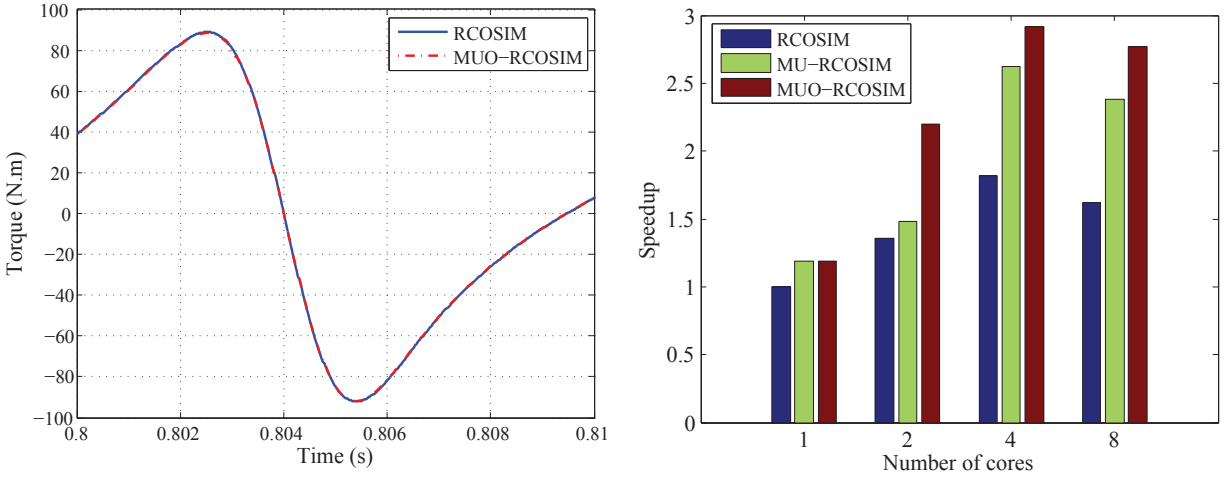
The proposed parallelization approach has been applied on an industrial use case. In this section, we give a description of the use case and then present the obtained results.

Tests have been performed on a computer running the Windows 7 operating system with 16 GB RAM and an Intel core i7 processor running 8 cores at 2.7 GH. Experiments have been carried out on a Spark Ignition (SI) RENAULT F4RT engine co-simulation. It is a four-cylinder in line Port Fuel Injector (PFI) engine in which the engine displacement is 2000 cm^3. The air path is composed of a turbocharger with a mono-scroll turbine controlled by a waste-gate, an intake throttle and a downstream-compressor heat exchanger. This co-simulation is composed of 5 FMUs: one FMU for each cylinder and one FMU for the airpath. The FMUs were imported into xMOD using the FMI export features of the Dymola tool. This use-case leads to an initial graph containing over 100 operations. We refer to our proposed method as MUO-RCOSIM (for Multi-Rate Oriented RCOSIM). We compared the obtained results with two approaches: The first one is RCOSIM which is mono-rate and thus we had to use the same communication step for all the FMUs. We used a communication step of $20\mu s$. The second one consists in using RCOSIM with the multi-rate graph transformation algorithm. We refer to it as MU-RCOSIM (for Multi-Rate RCOSIM). For MUO-RCOSIM and MU-RCOSIM we used the recommended configuration of the communication steps for this use case. For each cylinder, we used a communication step of $20\mu s$. The communication step used for the airpath is $100\mu s$. In fact, the airpath has slower dynamics than the cylinders and this configuration of the communication steps corresponds to the specification given by engine engineers. For each FMU, we used a Runge-Kutta 4 solver with a fixed integration step equal to the communication step. The graph of this use case is transformed by Algorithm 1 into a graph containing over 280 operations that are scheduled by the multi-core scheduling heuristic.

The validation of the numerical results of the co-simulation using the proposed method is achieved through the comparison of the co-simulation outputs with reference outputs. Since it is not possible to solve the equations of the FMU analytically, the reference outputs are obtained by using RCOSIM which has been shown in (Ben Khaled et al. 2014) to give a very good accuracy of the numerical results. Figure 5a shows the obtained results for the torque (an output of the airpath). We note that the results match with the reference, and the generated error is very small remaining within an acceptable bound ($< 1\%$). Similar accuracy results were obtained for the different outputs of the co-simulation.

The speedup obtained using MUO-RCOSIM is compared with the speedups obtained using RCOSIM and MU-RCOSIM. The speedup was evaluated by running the co-simulation in xMOD. Execution times measurements were performed by getting the system time stamp at the beginning and at the end of the co-simulation. For a given run of the co-simulation, the speedup is computed by dividing the mono-core co-simulation execution time of RCOSIM by the co-simulation execution time of this run on a fixed number of cores. Figure 5b sums up the results. The same speedup is obtained using MUO-RCOSIM and MU-RCOSIM even when only 1 core is used. This speedup is obtained thanks to using the multi-rate configuration. More specifically, increasing the communication step of the airpath from $20\mu s$ to $100\mu s$ results in fewer calls to the solver leading to an acceleration in the execution of the co-simulation. By using multiple cores, speedups are obtained using both MUO-RCOSIM and MU-RCOSIM. Additionally, MUO-RCOSIM outperforms MU-RCOSIM with an improvement in the speedup of, approximately 30% when 2 cores are used, and approximately 10% when 4 cores are used. This improvement is obtained thanks to the acyclic orientation heuristic which defines an efficient order of execution for the operations of each FMU that are mutually exclusive. This defined order tends to allow the multi-core scheduling heuristic to better adapt the

potential parallelism of the operation graph to the effective parallelism of the multi-core processor (number of cores) resulting in an improvement in the performance. MU-RCOSIM, on the other hand, uses the solution of RCOSIM which consists in simply allocating mutual exclusive operations to the same core introducing restrictions on the possible solutions of the multi-core scheduling heuristic. When using 8 cores, no further improvement is possible since the potential parallelism is fully exploited. Worse still, the overhead of the synchronization between the cores becomes counter-productive, which explains why the speedup with 8 cores is less than the speedup with 4 cores for all the approaches. The best performance is obtained using 5 cores with slight improvement compared to using 4 cores.



(a) Numerical results

(b) Speedup results

Figure 5: Speedup and numerical results of e-RCOSIM

7 CONCLUSION

This paper addressed the problem of automatic parallelization of multi-rate co-simulations using the FMI standard. We proposed an extension of the RCOSIM approach. First, we presented a set of rules and an algorithm in order to allow the execution of multi-rate FMUs where each FMU is executed according to its communication step, and correct data exchange between FMUs is ensured. Second, we presented an acyclic orientation heuristic that is applied on the operation graph in order to define an execution order between the operations of the same FMU which are mutually exclusive due to the non-thread-safe implementation of FMI. This heuristic results in a graph which can be efficiently exploited by the multi-core scheduling heuristic leading to an improvement in the co-simulation performance. The obtained speedup and numerical accuracy results show the efficiency of our proposed method. In future work, we aim at comparing our solution with an exact scheduling algorithm and also an online scheduling approach. Also, we will test our proposed method on other industrial applications. In addition, we envision to extend MUO-RCOSIM to real-time multi-core scheduling in order to perform Hardware-in-the-Loop simulation whose main challenge is to map the real-time constraints on the different operations of the graph.

REFERENCES

Al-Anzi, F. S., Y. N. Sotskov, A. Allahverdi, and G. Andreev. 2006. “Using Mixed Graph Coloring to Minimize Total Completion Time in Job Shop Scheduling”. *Applied Mathematics and Computation* vol. 182 (2), pp. 1137–1148.

Andreev, G. V., Y. N. Sotskov, and F. Werner. 2000. "Branch and Bound Method for Mixed Graph Coloring and Scheduling". In *Proceedings of the 16th International Conference on CAD/CAM, Robotics and Factories of the Future, CARS and FOF*, pp. 1–8.

Ben Khaled, A., M. Ben Gaid, N. Pernet, and D. Simon. 2014. "Fast Multi-core Co-simulation of Cyber-Physical Systems: Application to Internal Combustion Engines". *Simulation Modelling Practice and Theory* vol. 47, pp. 79–91.

Blochwitz, T., M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel et al. 2012, September. "Functional Mockup Interface 2.0: The Standard for Tool Independent Exchange of Simulation Models". In *Proceedings of the 9th International Modelica Conference*, Number 076, pp. 173–184. Munich, Germany, Linköping University Electronic Press.

Elmqvist, H., S. Mattsson, and H. Olsson. 2014. "Parallel Model Execution on Many Cores". In *Proceedings of the 10th International Modelica Conference*. Lund, Sweden.

Galtier, V., S. Vialle, C. Dad, J.-P. Tavella, J.-P. Lam-Yee-Mui, and G. Plessis. 2015. "FMI-based Distributed Multi-Simulation with DACCOSIM". In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pp. 39–46. Society for Computer Simulation International.

Gebremedhin, M., A. H. Moghadam, F. Fritzson, and K. Stavaker. 2012. "A Data-Parallel Algorithmic Modelica Extension for Efficient Execution on Multi-Core Platforms". In *Proceddings of the 9th International Modelica Conference*. Munich, Germany.

Grandpierre, T., C. Lavarenne, and Y. Sorel. 1999, May. "Optimized Rapid Prototyping For Real-Time Embedded Heterogeneous multiprocessors". In *Proceedings of the 7th International Workshop on Hardware/Software Co-Design, CODES'99*. Rome, Italy.

Ramamritham, K. 1995, April. "Allocation and Scheduling of Precedence-Related Periodic Tasks". *IEEE Transactions on Parallel and Distributed Systems* vol. 6 (4), pp. 412–420.

Saidi, S. E., N. Pernet, Y. Sorel, and A. Ben Khaled. 2016, May. "Acceleration of FMU Co-Simulation On Multi-core Architectures". In *Proceedings of the First Japanese Modelica Conference*. Tokyo, Japan.

Sirin, G., C. J. Paredis, B. Yannou, E. Coatanéa, and E. Landel. 2015. "A Model Identity Card to Support Simulation Model Development Process in a Collaborative Multidisciplinary Design Environment". *IEEE Systems Journal* vol. 9 (4), pp. 1151–1162.

Sotskov, Y. N., V. S. Tanaev, and F. Werner. 2002. "Scheduling Problems and Mixed Graph Colorings". *Optimization* vol. 51 (3), pp. 597–624.

AUTHOR BIOGRAPHIES

SALAH EDDINE SAIDI is a PhD student at Pierre et Marie Curie University and holds a Masters in Computer Engineering from the same university. As part of the PhD program he is a research scholar at IFP Energies nouvelles and INRIA. His research interests lie in co-simulation parallelization, multi-core, embedded, and real-time computing. His email address is salah-eddine.saidi@ifpen.fr.

NICOLAS PERNET is a Software Project Manager at IFP Energies nouvelles. He holds a PhD in Computer Science from Pierre et Marie Curie University. His research interests include real-time systems, multi-core programming paradigms, and co-simulation approaches. His email address is nicolas.pernet@ifpen.fr.

YVES SOREL is Research Director at Inria, the French National Institute for computer science and applied mathematics, and scientific leader of the team AOSTE-Paris. His main research topic concerns the analysis and the optimization of distributed real-time embedded systems. He is also the founder of the AAA methodology (Algorithm Architecture Adequation) and of SynDEx, a system level CAD software for distributed real-time embedded systems. His email address is yves.sorel@inria.fr.