Parallelization Approaches for the Time-Efficient Simulation of Hybrid Dynamical Systems: Application to Combustion Modeling

Abir Ben Khaled, Mohamed El Mongi Ben Gaïd, Daniel Simon

Parallelization Approaches for the Time-efficient Simulation of Hybrid Dynamical Systems: Application to Combustion Modeling

Abir Ben Khaled[1] *Mongi Ben Gaid*[1] *Daniel Simon*[2]

[1]IFP Energies nouvelles, 1-4 avenue de Bois-Préau, 92852 Rueil-Malmaison, France,
{abir.ben-khaled, mongi.ben-gaid}@ifpen.fr
[2]INRIA and LIRMM-CNRS-Université Montpellier Sud de France, DEMAR team, 95 rue de la Galéra,
34090 Montpellier, France, daniel.simon@inria.fr

Abstract

The need for time-efficient simulation is increasing in all engineering fields. Potential improvements in computing speeds are provided by multi-core chips and parallelism. However, the efficient numerical integration of systems described by equation oriented languages requires the ability to exploit parallelism. This paper investigates the problem of the efficient parallelization of hybrid dynamical systems both through the model and through the solver. It is first argued that the parallelism is limited by dependency constraints between sub-systems, and that slackened synchronization between parallel blocks may provide speed-ups at the cost of induced numerical errors, which are theoretically examined. Then two methods for automatic block diagonalization are presented, using bipartite graphs and hypergraphs. The application of the latter method to hybrid dynamical systems, both from the continuous state variables and discontinuities point or view, is investigated. Finally, the model of a mono-cylinder engine is analyzed from equations point of view and a possible split using the hypergraph method is presented and discussed.

Keywords Parallel computing, model decomposition, delay error, dependencies constraints, multicore simulation

1. Introduction

The design and validation of complex systems, like cyber-physical systems which include both physical and computational devices, is costly, time consuming and needs knowledge and cooperation of different disciplines [15]. For example, vehicle power-trains belong to such category and require the coordinated design of both mechanical, electrical, thermodynamic and chemical models (from a physical point of view) and many-sided controllers (from a computational point of view).

A global simulation is needed at an early stage to speed-up the design, development and validation phases. The main purpose of the numerical simulation is, when an analytic solution cannot be derived, to approximate as faithfully as possible the behavior of the complex dynamic system. In other words, bounding and minimizing the simulation errors is an important goal of numerical simulations so that the designers can be confident with the prediction.

Simulating complex systems is time consuming in term of calculations, and reaching real-time is often out of the capabilities of single processors. Parallel computing can be performed by splitting the models into several sub-models that are concurrently simulated on several processors to ensure the compliance with the real-time constraints.

The data dependencies due to the coupled variables between sub-models lead for waiting periods and idle processor time, decreasing the efficiency of threaded parallelism on the multicore platform. Therefore these dependency constraints should be relaxed as far as possible. However, to avoid too large numerical errors in the simulation results, a minimal synchronization between sub-models must be achieved, and the parallelism between computations must be carefully restricted.

The relaxation of the synchronization constraints, while guaranteeing correct simulation results, needs to split the model properly before distribution over several CPUs. An efficient decomposition relies on knowing how and where to cut in order to decouple subsystems as far as possible. Relaxed data dependencies may lead to slack synchronization between sub-models, until reaching an acceptable trade-off between the computation costs and the simulation precision.

The aim of this paper is to propose approaches for the time-efficient and real-time simulation of hybrid dynamical systems, showing two techniques of parallelization which can be combined to reach complementary objectives:

- Parallelization across the model where the delay error due to the model decomposition is evaluated to determine the involvement and the weighting of each of its elements, and consequently to know how to act for the error reduction. The parallelization approach is based on a system

5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. 19 April, 2013, University of Nottingham, UK.
 Proceedings available at:
http://www.ep.liu.se/ecp_home/index.en.aspx?issue=084
EOOLT 2013 website:
http://www.eoolt.org/2013/

splitting using the block-diagonal forms of state and event incidence matrices.

- Parallelization through the solver where a new method of parallelization at the event detection and location level is presented. The parallelization approach is situated on the solver level using the block-diagonal form of the event incidence matrices.

This paper is organized as follows. First, a formal model of a hybrid dynamic system is established. After that, delay errors due to parallelization across the model are evaluated in the context of real-time simulation. Then a study of system splitting using the block-diagonal form of incidence matrices, related to states and events, is performed. Finally, a case study concerning a mono-cylinder engine model is presented and test results are performed by analyzing the relationships between the states, the events and the states/events together.

2. Related Work

In the context of the co-simulation or the parallelization across the model, where the system is split into several sub-systems and simulated in parallel on different processors, several works already targeted the real-time distributed simulation of complex physical models. In [9], the study focused on the case of fixed-step solvers. Then, in [2], the study was extended to examine the case of variable-step solvers. Besides, in [16] distributed simulation was performed in Modelica [11], using a technology based on bilateral delay lines called transmission line modeling (TLM) combined with solver in-lining. The TLM technique consider the decoupling point when the variables change slowly and the time-step of the solver is relatively small, so that the decoupled subsystems can be considered as connected by constant variables.

Another kind of parallelization is called parallelization through the solver. It concerns the execution of the model on a single thread while its numerical resolution is parallelized. It relies on using well know parallel approaches for solving the required steps of the used solver. For example, parallelizing matrix inversions, which are needed when using an implicit method (see [17] and [12]) and parallelizing operations on vectors for ODEs resolution by separating them into modules (see PVODE in [7] implemented using MPI (Message-Passing Interface) technology).

3. Problem Formalization

In this kind of cyber-physical systems the physical part is modeled in the continuous-time domain using hybrid ODEs. It belongs to the hybrid systems category because of some discontinuous behaviors, that correspond to events triggered off when a given threshold is crossed. Controllers, which interact with physical parts, represent computational models. They are modeled on the discrete-time domain and sampling is a mixture of time-driven and events-driven features.

Let us provide a formal model, considering a hybrid dynamic system Σ whose continuous state evolution is governed by

$$\dot{X} = f(t, X, D, U) \quad \text{for } t_n \leq t < t_{t+1}, \tag{1a}$$

$$Y = g(t, X, D, U). \tag{1b}$$

where $X \in \mathbb{R}^{n_X}$ is the continuous state vector, $D \in \mathbb{R}^{n_D}$ is the discrete state vector, $U \in \mathbb{R}^{n_U}$ the input vector, $Y \in \mathbb{R}^{n_Y}$ is the output vector and $t \in \mathbb{R}^+$ is the time.

$(t_n)_{n\geq 0}$ is a sequence of strictly increasing time instants representing discontinuity points called "state events", which are the roots of the equation

$$h(t, X, D, U) = 0. \tag{2}$$

h is usually called zero-crossing function or event indicator, used for *event detection* and *location* [18].

At each time instant t_n, a new continuous state vector may be computed as a result of the *event handling*

$$X(t_n) = I(t_n, X, D, U), \tag{3}$$

and a new discrete state vector may be computed as a result of discrete state update

$$D(t_n) = J(t_{n-1}, X, D, U), \tag{4}$$

If no discontinuity affects a component of $X(t_n)$, the right limit of this component will be equal to its value at t_n.

This hybrid system model is adopted by several modeling and simulation environments and is underlying the functional mock-up interface (FMI) specification [6].

We assume that Σ is well posed in the sense that a unique solution exists for each admissible initial conditions $X(t_0)$ and $D(t_0)$ and that consequently X, D, U, and Y are piecewise continuous functions in $[t_n, t_{n+1})$.

The system must be split for numerically integration on a set of $\mathcal{P}$ cores. The objective is to speedup the simulation through the exploitation of parallelism, while keeping the following objectives in mind [2]:

- minimization of the delay errors, which are related both to the dependency constraints and to the distribution of the state variables;
- optimization of the computational resource usage through load balancing and idle time avoidance: high CPU utilization is influenced by the size and independence of the computational tasks;
- minimization of the number of solvers interrupts due to discontinuities unrelated with the local sub-system: discontinuities need that the corresponding numerical solver must be stopped and restarted, therefore introducing computation overheads;
- keeping an acceptable level of numerical stability and accuracy.

4. Evaluation of Delay Errors due to Decoupling

4.1 Context and Motivation

To begin with something simple, assume now that the system can be split into two subsystems as in Figure 1.

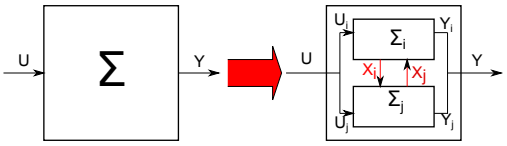

Figure 1. System splitting for parallelization

For simplicity, the analysis is focused only on continuous time-invariant models. In the following, we will denote by $x(n) = x(t_n)$. Therefore, the system can be written as:

$$\begin{cases} \dot{x}_i &= f_i(x_i, x_j, U_i) \\ Y_i &= g_i(x_i, x_j, U_i) \end{cases} \quad and \quad \begin{cases} \dot{x}_j &= f_j(x_j, x_i, U_j) \\ Y_j &= g_i(x_j, x_i, U_j) \end{cases} \tag{5}$$

with $X = [x_i \;\; x_j]^T$

Here U_i are the inputs needed for $\sum_i$ and U_j are the inputs needed for $\sum_j$. In other words, $U_i \cup U_j = U$ and $U_i \cap U_j$ can be an empty set or not according to the achieved decoupling.

In the same way, Y_i are the outputs produced by $\sum_i$ and Y_j are the outputs produced by $\sum_j$. In other words, $Y_i \cup Y_j = Y$ and $Y_i \cap Y_j = \oslash$.

After parallelization, as shown in Figure 1, each subsystem Σ_i have a private subset U_i as input vector and a private subset Y_i as output vector, and the direct interaction between subsystems are only due to a subset made of variables of the state vector. Hence the focus is now on the internal data flow, i.e. the shared state variables to be identified, and whose number should be minimized to enhance the independence of parallel branches.

In general, in order to compute the next state $x_i(n+1)$, numerical integrators use [1, 8]:

- Always, the current value of the state $x_i(n)$ and the derivative $f_i(X(n))$.
- For multi-step methods, the m previous values of the state $x_i(n-\alpha)$ and/or the derivatives $f_i(X(n-\alpha))$ (with $\alpha = 1, .., m$).
- For methods with order higher than one, higher derivative such as $f_i(X(n) + \beta.f_i(X(n)))$ for a 2^{nd} order.
- For implicit methods, the future value of the derivative $f_i(X(n+1))$ computed through iterations.

Consequently, in order to compute the next state value $x_i(n+1)$, the numerical solver needs at least values for $x_i(n)$ and $\dot{x}_i(n) = f_i(X(n)$ (see Figure 2).

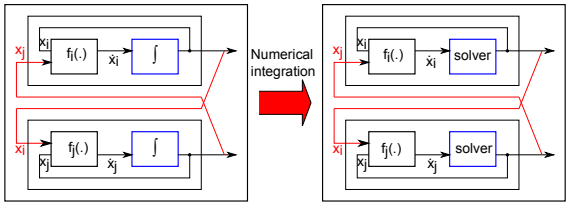

Figure 2. System's internal composition

However, when the computation of $\dot{x}_i(n) = f_i(X(n))$ is required, the value of $x_i(n)$ is already available, but this assertion is invalid for $x_j(n)$ with $j \neq i$. In fact, x_j is only updated every synchronization interval P that is larger than the integration step. In other words, $x_j(n)$ can be available only if the time t_n corresponds to the synchronization time, otherwise its value will be the one of last updated. Thereby, the focus will be then in $\dot{x}_i(n) = f_i(x_j(n))$ for $j \neq i$.

4.2 Evaluation of Delay Errors in a Basic Problem

In order to evaluate the influence of using a delayed value of x_j in $f_i(.)$ due to the lack of communication at processing time, let us start by a simple case with a single state variable x. The analysis and results carried out in this case will be further applied to the large problem with N state variables $X = [x_1 \ldots x_N]^T$. Therefore, a backup of x correspond to a synchronization between dependent variables x_i and x_j where $j \neq i$.

Assume that x is integrated every step h_k. As a first case, we consider that the value of the state variable x is saved every integration step in order to be available for the next computation. As a second case, we just consider the availability of an old saved (delayed) value at $n - \alpha$.

$$\textbf{Case 1:} \quad x(n+1) = x(n-\alpha) + \sum_{k=n-\alpha}^{n} h_k.\dot{x}(k) \tag{6}$$

$$\textbf{Case 2:} \quad \tilde{x}(n+1) = x(n-\alpha) + \sum_{k=n-\alpha}^{n} h_k.\dot{x}(n-\alpha) \tag{7}$$

The error resulted from the difference between (6) and (7) (see Figure 3) is represented by e:

$$\begin{aligned} e(n+1) &= x(n+1) - \tilde{x}(n+1) \\ &= \sum_{k=n-\alpha+1}^{n} h_k.(\dot{x}(k) - \dot{x}(n-\alpha)) \\ &= e(n) + h_n.(\dot{x}(n) - \dot{x}(n-\alpha)) \end{aligned} \tag{8}$$

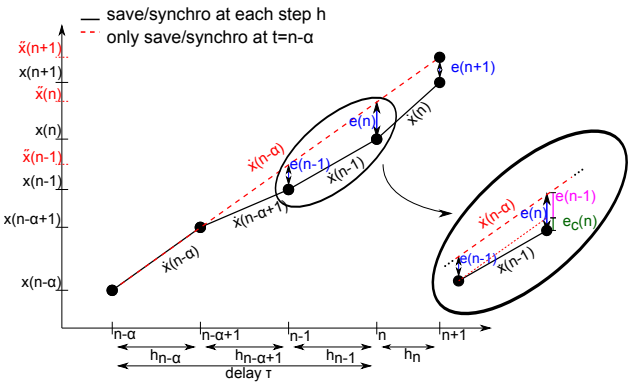

Figure 3. Errors due to delays

To show up the delay τ in the expression of e, (8) is rewritten in the temporal form

$$e(t_{n+1}) = e(t_n) + h_n.(\dot{x}(t_n) - \dot{x}(t_n - \tau)) \text{ where } \tau = t_n - t_{n-\alpha} \tag{9}$$

Let $e_c(t_{n+1}) = h_n.(\dot{x}(t_n) - \dot{x}(t_n - \tau))$. Here $e_c(t_{n+1})$ is the current error generated at t_{n+1}. So, the final error $e(t_{n+1})$ is the result of the accumulation of a past error $e(t_n)$ and the current error $e_c(t_{n+1})$. As a conclusion, two conditions must be met to achieve a correct result:

- $e_c(t_{n+1}) < \epsilon_{loc}$: allowed local error
- $e(t_{n+1}) < \epsilon_{glo}$: allowed global error

4.3 Evaluation of Delay Errors in a General Problem

The analysis and results made in the previous case (the basic problem) will be applied to the following larger problem with N state variables $X = [x_1 \dots x_N]^T$.

Assume that a given system Σ is split to two separated subsystems Σ_A and Σ_B with state vectors X_A and X_B such as $X_A \cup X_B = X$ and $X_A \cap X_B = \oslash$. Then assume that they are integrated with a variable time-step h_k for X_A and $h_{k'}$ for X_B. Besides, they are synchronized every period P, where $P >> h_k$, in order to exchange some updated data.

The aim is to find an evaluation of the error, caused by the lack of an updated data during a delay τ, at any given time t_k for X_A and $t_{k'}$ for X_B, which may be a synchronization time or to an integration time between two checkpoints.

Thereby, the delay τ is represented by the difference between the current integration time t_k and the last synchronization time t_s as follow

$$\tau = t_k - t_s \tag{10}$$

where t_s is computed as follow

$$t_s = P.\left\lfloor \frac{t_k}{P} \right\rfloor \tag{11}$$

in order to have

$$t_s = \begin{cases} l.P & when \quad t_k = l.P \quad l \in \mathbb{N}^* \\ (l-1).P & when \quad t_k < l.P \quad l \in \mathbb{N}^* \end{cases} \tag{12}$$

which leads to

$$\begin{cases} \tau = 0 & when \quad t_k = t_s \\ \tau > 0 & when \quad t_k > t_s \end{cases} \tag{13}$$

Therefore, the resulted error at t_{n+1} in the subsystem X_A denoted by $E_A(t_{n+1})$ will be the difference between $X_A(t_{n+1})$ and $\tilde{X}_A(t_{n+1})$ deduced from (14) and (15).

$$X_A(t_{k+1}) = X_A(t_k) + h_k.f_A(X_A(t_k), X_B(t_k)), \quad k \in \{0, \dots, n\} \tag{14}$$

$$\tilde{X}_A(t_{k+1}) = \begin{cases} X_A(t_{k+1}) & k = 0 \\ \tilde{X}_A(t_k) + h_k.f_A(\tilde{X}_A(t_k), \tilde{X}_B(t_k - \tau)) & k \geq 1 \end{cases} \tag{15}$$

In other words,

$$\begin{aligned} E_A(t_{n+1}) &= \sum_{k=0}^{n} E_A(t_k) \\ &+h_n.[f_A(X_A(t_n), X_B(t_n)) - f_A(\tilde{X}_A(t_n), \tilde{X}_B(t_n - \tau))] \\ &= E_{A,p}(t_n) + E_{A,c}(t_{n+1}) \end{aligned} \tag{16}$$

where

$$\begin{aligned} E_{A,c}(t_{n+1}) &= h_n.[f_A(X_A(t_n), X_B(t_n)) - f_A(\tilde{X}_A(t_n), \tilde{X}_B(t_n - \tau))] \\ E_{A,p}(t_n) &= \sum_{k=0}^{n} E_A(t_k) \end{aligned} \tag{17}$$

Here $E_{A,c}(t_{n+1})$ is the current error generated at t_{n+1} whatever a synchronization or not. So, the global decoupling error $E_A(t_{n+1})$ is the result of the accumulation of a past errors $E_{A,p}(t_n)$ and the current error $E_{A,c}(t_{n+1})$.

As a conclusion, to achieve a correct result, two conditions must be met for the current (local) error and the global error:

- $E_{A,c}(t_{n+1}) < \epsilon_{loc}$
- $E_A(t_{n+1}) < \epsilon_{glo}$

These conditions can be satisfied by acting on some parameters. In fact, in (17), the delay error depends on the integration steps h_k and on the delay τ. The delay τ depends on the last synchronization time t_s, which itself depends on the synchronization period P, and on the current integration time t_k. In other word, the delay error depends on the size and number of integration steps since the last synchronization. Indeed, the step size gives an idea of the sharpness of the state variations, and numerous small integration steps denotes large variations between successive updates. Fast variations of the state variables are not only related to the system stiffness, but also especially to the presence of discontinuities. Therefore delay errors are strongly related with discontinuities, whose location and dependencies deserve to be carefully examined when splitting the system.

In addition, the delay error size increases with the number of dependencies between the subsystems. Obviously, if X_A is independent from X_B, no synchronization error can be generated. If conversely X_A depends on all the state variables of X_B, the errors may increase considerably. Thus the objective is to choose cuts in the system to minimize the data exchanges and to control the growth of the integration errors.

4.4 Delay Errors in a Real-time Simulation

Up-to-now, the delay errors are evaluated in the case where the integration of all subsystems was exactly finished at the synchronization point. In fact we consider that the synchronization points are deadlines, without worrying about what happens inside at each integration step. The previous study just treated the case of hard real-time constraints where all the deadlines are met.

To speed-up the simulation, waiting periods should be eliminated. These idle times occur at the synchronization points when one or more sub-systems are waiting for the end of integration of the other sub-systems. Therefore, this time can be decreased by using slackened synchronization [5]. It means that, at the synchronization points, a subsystems which needs variable does not wait for its integration to be completed, but uses the last available value. Indeed this relaxation induces a cost in term of delay error, which must be re-evaluated to check the trade-off between simulation speed and accuracy.

Assuming that only the last integration step before the synchronization can be missed, the value and production time of the last completed integration step before the synchronization, (denoted t_p for X_A and $t_{p'}$ for X_B) must be known: $t_p = t_s - h_p$.

Assume that the synchronization deadlines are always missed, so that the error E_A is always computed from the

last completed integration step. Thereby, the delay τ computed beforehand in (10) is no longer valid but is as follow:

$$\tau = t_k - t_p \tag{18}$$

In other words, instead of $\tilde{X}_B(t'_k - \tau) = \tilde{X}_B(t_s)$, now $\tilde{X}_B(t'_k - \tau) = \tilde{X}_B(t'_p)$ leading to

$$\begin{aligned} E_{A,c}(t_{n+1}) &= h_n.[f_A(X_A(t_n), X_B(t_n)) - f_A(\tilde{X}_A(t_n), \tilde{X}_B(t'_p))] \\ E_{A,p}(t_n) &= \sum_{k=1}^{n} E_A(t_k) \end{aligned} \tag{19}$$

Here the two equations (17) and (19) are equivalent, the only difference lies in the expression of τ.

5. System Splitting Using Block-diagonal Forms

As mentioned before, the purpose is to optimize the exploitation of the parallelism of the sub-systems while keeping the previously evaluated delay error due the decoupling under control. Two methods have been analyzed for this aim, the first is related to the states to reduce the data-flow due to coupling variables between sub-systems. The second one is related to the events, to reduce integration interrupts, and also to minimize event detection and location via a complementary kind of parallelization through the solver.

5.1 Accounting for the State Variables

To reduce the data exchange between two sub-models and to prioritize these swaps inside one sub-model, the dependencies between the state variables must be evaluated. It can be done either by a direct access to the incidence matrix that describes the coupling between the state variables and their derivatives, or by computing the Jacobian matrix.

A Jacobian matrix is a matrix of all first-order partial derivatives of a vector function $f = [f_1 f_2 \dots f_N]^T$ regarding another vector $X = [x_1 x_2 \dots x_N]^T$. An $N \times N$ Jacobian matrix denoted by J has the form:

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_N}{\partial x_N} \end{pmatrix}$$

If there is a zero element in the Jacobian, i.e. $\frac{\partial f_i}{\partial x_j} = 0$, it means that f_i is not influenced by x_j. However, f_i is actually $\dot{x}_i$. In other words, x_i does not depend on x_j. In the same way if $\frac{\partial f_i}{\partial x_j} \neq 0$, it means that x_i depends on x_j. Moreover, the numerical value of $\frac{\partial f_i}{\partial x_j}$ gives a measure of the sensitivity of $\dot{x}_i$ w.r.t. x_j.

This leads to conclude that the Jacobian matrix can be seen as an incidence matrix which provides useful information about data dependencies between state variables. This could be used for an effective system splitting. So that, when transforming the matrix into a block-diagonal form by permuting rows and columns, the blocks represents the independent subsystems. It may happen that a total block-diagonalization is not possible so that the final transformed matrix presents some coupling rows and/or column, this denotes the presence of irreducible dependencies between subsystems.

5.2 Accounting for the Discontinuities

To minimize the delay error while optimizing the exploitation of the parallelism across the model, it is also crucial to reduce the number of discontinuities inside each sub-model, so that stiff variations of the state variables are limited. This procedure induces another benefit, as reducing the number of interrupts for each solver reduces re-starting overheads and improves the integration speed.

The events incidence matrix describes the relationships between events. Block-diagonalizing this matrix allows for separating the discontinuities and scatter them in the different sub-models.

Furthermore, the events incidence matrix block-diagonalization also leads to a kind of parallelization across the solver. In fact, the system resolution, including events handling, consists of 4 steps as mentioned in Figure 4.

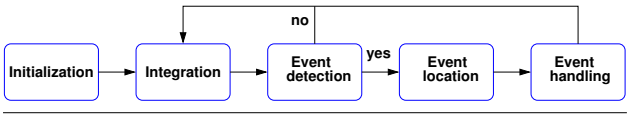

Figure 4. Events handling operations flow

Event detection and location can be an expensive stage for hybrid systems (and for the addressed combustion models in particular). Especially, the event location (i.e. solving the zero-crossing equation (2)) can take a long time through an iterative process, and it is difficult to bound this step. By using the event incidence matrix, solving for a particular event can be localized in a subset of the global system through parallelization, thus shortening the zero-crossing function solving.

5.3 Permuting Sparse Rectangular Matrices for Block-diagonal Forms

Two methods and associated software tools have been evaluated to perform the system diagonalization. Note that the original state variables of the system are preserved and that diagonal forms are produced only through permutations.

5.3.1 Bipartite Graph Model

A matrix A is transformed to a bipartite graph model. This graph is used by a specific tool to partition it, then to get a doubly bordered block-diagonal matrix A_{DB}, i.e. the matrix has a block-diagonal form with non-zero elements on its last rows and columns as in Figure 5.

MeTiS [13] is a software aimed to partition large graphs. The used algorithms are based on multilevel graph partitioning, which means reducing the size of the graph by collapsing vertices and edges, then partitioning the smaller graph, and finally uncoarsening it to construct a partition for the original graph.

The block-diagonal form is performed by permuting rows and columns of a sparse matrix A to transform it into a K-way doubly bordered block-diagonal (DB) form A_{DB}. It has a coupling row and a coupling column.

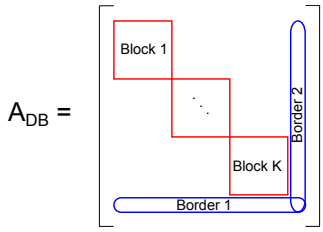

Figure 5. Doubly bordered block-diagonal matrix

The representation of the nonzero structure of a matrix by a bipartite graph model reduces the permutation problem to those of graph partitioning by vertex separator (GPVS).

For example, let A the following matrix:

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (20)$$

An undirected graph $G = (V, E)$ is defined as a set of vertices V and a set of edges E. The corresponding bipartite graph for MeTiS is built by replacing the rows and the columns by vertices and the non-zeros are represented by edges. After transformation, MeTiS partitions the graph as shown in Figure 6.

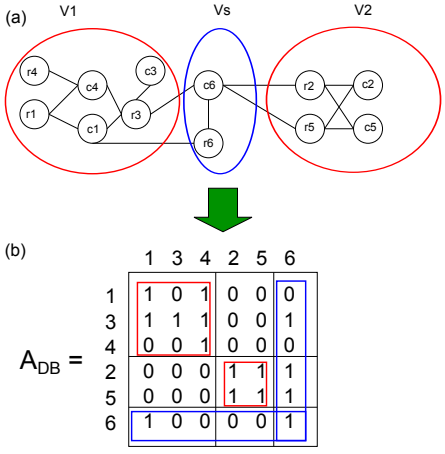

Figure 6. (a) Bipartite graph representation of the matrix A and 2-way partitioning of it by vertex separator Vs; (b) 2-way DB form of A induced by (a)

The objective of MeTiS when partitioning is to:

- Minimize the size of the separator because it implies the minimization of the border size.
- Balance among sub-bipartite graphs because it implies a balance among diagonal sub-matrices.

5.3.2 Hypergraph Model

A matrix A is transformed to a hypergraph model. An hypergraph $H = (U, N)$ is defined as a set of nodes (vertices) U and a set of nets (hyper-edges) N among those vertices. This hypergraph is used by a specific tool to partition it, then to get a singly bordered block-diagonal matrix A_{SB} as in Figure 7, where the matrix has a block-diagonal form with non-zero elements only on its last rows.

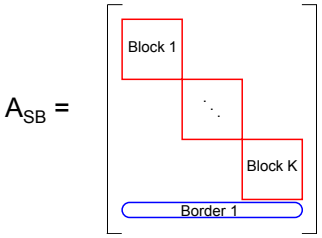

Figure 7. Singly bordered block-diagonal matrix

PaToH (Partitioning Tools for Hypergraphs) [19] is a multilevel hypergraph partitioning tool that consist of 3 phases as for MeTiS: coarsening, initial partitioning, and uncoarsening. In the first phase, a multilevel clustering, that correspond to coalescing highly interacting vertices to super-nodes, is applied on the original hypergraph by using different matching heuristics until the number of vertices drops below a predetermined threshold value. Then, the second phase corresponds to partition the coarsest hypergraph using diverse heuristics. Finally, in the third phase, the obtained partition is projected back to the original hypergraph by refining the projected partitions using different heuristics.

The block-diagonal form is performed by permuting rows and columns of a sparse matrix A in order to transform it into a K-way singly bordered block-diagonal (SB) form A_{SB}. It has only a coupling row. For this reason, this method of block-diagonalization will be selected for the later study.

The representation of the nonzero structure of a matrix by an hypergraph model reduces the permutation problem to those of hypergraph partitioning (HP).

The corresponding hypergraph graph of the matrix A (20) for PaToH is build by replacing the rows and the columns of the matrix by nets and nodes respectively. The number of pins is equal to the number of non-zeros in the matrix. After the transformation, PaToH partitions the hypergraph as as it is shown in Figure 8.

The objective of PaToH when partitioning is to:

- Minimize the cut size because it implies the minimization of the number of coupling rows.
- Balance among sub-hypergraphs because it implies a balance among diagonal sub-matrices.

In conclusion, the method using the bipartite graph model as MeTiS generates a doubly bordered block-diagonal matrix. To further reduce the coupling row and the coupling column to a single coupling row, the Ferris-Horn (FH) algorithm [10] uses a column splitting method. Unfortunately, the number of rows and columns of the matrix must be increased. In contrast, the method using the hypergraph model as PaToH directly generates a singly bordered block-diagonal matrix which means only a coupling row without adding an intermediate method. Therefore PaToH

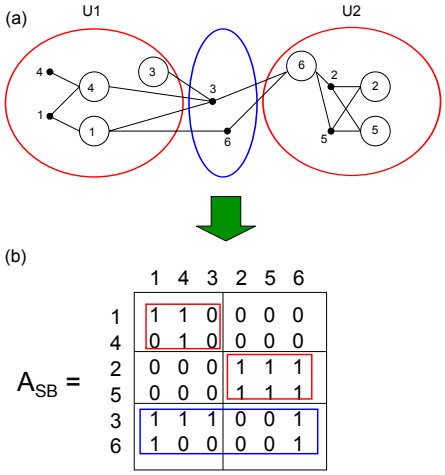

Figure 8. (a) Row-net hypergraph representation of the matrix A and 2-way partitioning of it; (b) 2-way SB form of A induced by (a)

will be used for the block-diagonalization of matrices in the following case study.

6. Analysis of System Splitting using an Hypergraph Model through a Case-study

In this study, a Spark Ignition (SI) mono-cylinder engine has been modeled (see Figure 9) with 3 gases (air, fuel and burned gas). It was developed using the ModEngine library [3]. ModEngine is a Modelica [11] library that allows for the modeling of a complete engine with diesel and gasoline combustion models. It contains more than 250 sub-models. A variety of elements are available to build representative models for engine components. ModEngine is currently functional in the Dymola tool [1].

6.1 Engine Modeling

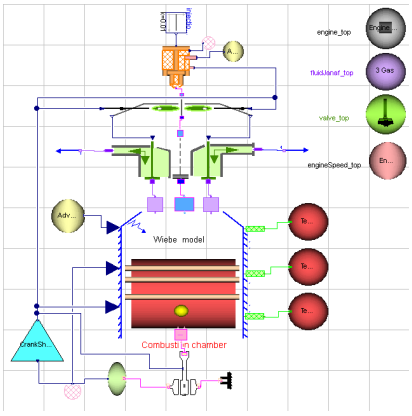

Figure 9. Mono-cylinder engine modeled in Dymola

In the following tests, relationships between state variables and events as well as their behaviors are essential to study how to split the system at wisely chosen joints.

For this aim, the mono-cylinder model, written in Modelica language, was translated to a simpler language called Micro-Modelica (μ-Modelica) [4], which is understandable by the stand-alone Quantized State Systems (QSS) tool [14] as shown in Figure 10. The QSS solver is not used here, only a related tool is used to generate a so-called simulation file which contains important information about the system and relationships between states and events. These data are extracted thereafter by a custom dedicated tool, and translated both to a matrix form for visualization and to an hypergraph file for the PaToH tool. Finally PaToH generates a partitioned hypergraph file that describes how the graph is decomposed and transformed subsequently to a matrix form for visualization.

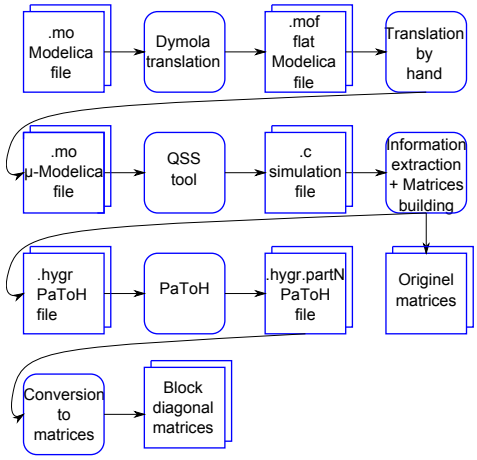

Figure 10. Software tool-chain

The considered mono-cylinder model is characterized by a number of:

- State variables: $n_X = 15$
- Events: $n_Z = 111$
- Discrete variables: $n_D = 93$

The state variables x_i $(i = 1, .., n_X)$ are defined as follows:

ID	Name	Details
X_0	CrankAngle	Crank Shaft angle
$X_{1\to3}$	mEvapo[3]	Gas mass evaporated due to injection in global Mass balance equation
X_4	qvAlfa	Current released heat generated by the combustion process
X_5	mrefAlfa	Burned mass fraction during combustion process
X_6	combuHeatRelease	Output current combustion heat released
$X_{7\to9}$	mCombu[3]	Gas mass derivatives due to combustion in global Mass balance equation
$X_{10\to12}$	M[3]	Mass of gas
X_{13}	Energy	Energy contained in the cylinder
X_{14}	cylinderTemp	Output temperature in the cylinder

[1] http://www.3ds.com/products/catia/portfolio/dymola

The events z_i ($i = 1, .., n_Z$) and discrete variables d_i ($i = 1, .., n_D$) are defined by the "when" blocks as follows:

```
when (z_i) then
 d_i = ... ;
elsewhen !(z_i) then
 d_i = ... ;
end when;
```

The statements that are between the "then" and the "elsewhen" or the "end when" are called the event handler, it represents the consequence of the event.

6.2 State and Derivatives Incidence Matrices

At first glance, the number of coupled state variables is 6 among 15. In fact, $\dot{X}_{13}$ is only influenced by the state variables X_0, X_{10}, X_{11}, X_{12}, X_{14} as shown in Figure 11.

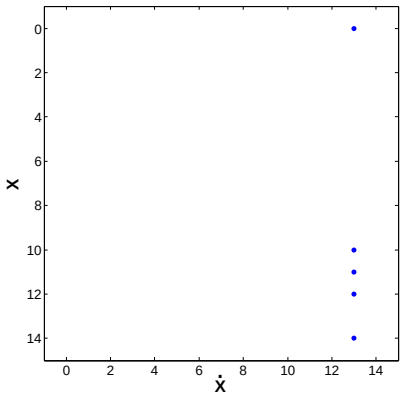

Figure 11. Incidence state matrix: derivatives of state variables $\dot{X}$ depending on state variables X

Thus far, considering only the incidence state matrix, only 40% of the state variables are directly computed from the other states, while the others depend on external inputs (or even remain constant on some particular trajectories of the state space, e.g. when imposing a constant velocity of the crank).

The same result is found for events. In fact, the number of active events is 39 among 111, as the previous cited involved state variables directly affect values of 39 events as shown in Figure 12.

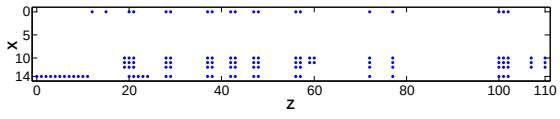

Figure 12. Incidence matrix: events Z depending on the state variables X

This number represents only 35% of the total number of events, while the rest is only used to activate other events. In fact these 72 events are defined in the ModEngine library to be used in more general systems, not for the particular mono-cylinder use case. In consequence only the subset of active events must be detected.

However, if the state variables X can affect the events Z, the events can also change the state variables values. In order to construct its corresponding matrix, both the incidence matrix that defines the discrete variables D influenced by the events Z: $Z \rightarrow D$ (see Figure 13) and the incidence matrix that defines the derivatives of the state variables $\dot{X}$ influenced by the discrete variables D: $D \rightarrow \dot{X}$ (see Figure 14) are carried out.

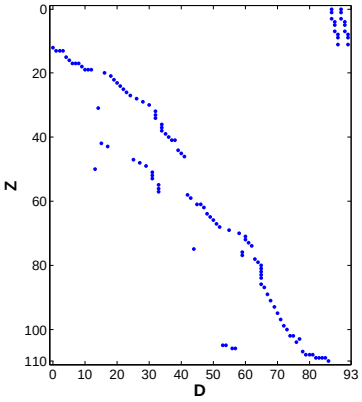

Figure 13. Incidence matrix: discrete variables D influenced by the events Z

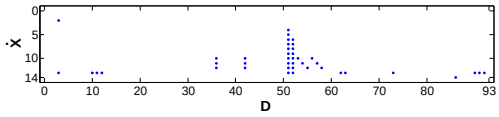

Figure 14. Incidence matrix: derivatives of state variables $\dot{X}$ influenced by discrete variables D

Thus the incidence matrix $Z \rightarrow \dot{X}$ is deduced by transitivity in Figure 15.

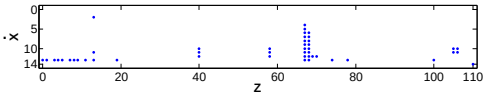

Figure 15. Incidence matrix: derivatives of state variables $\dot{X}$ influenced by the events Z

Figure 15 shows that the previous identification of some state variables as not coupled (based on incidence state matrix Figure 11) and events influenced by state variables (Figure 12), is no longer true. In fact, now 13 state variables among 15 appear in this incidence matrix. Note that now only the state variables corresponding to X_1 and X_3 do not appear in this incidence matrix, this is due to the fact that these variables are inhibited momentarily to test a particular scenario.

By combining the two matrices in Figures 12 and 15, an incidence matrix between events and state variables can be achieved as in Figure 16.

Once unnecessary states and events are eliminated and only involved ones are kept, the intrication between state

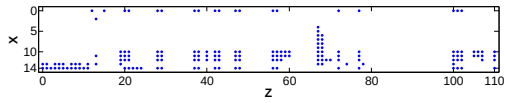

Figure 16. Incidence matrix: data exchange between events Z and state variables X

variables and events in both directions shows that it is difficult to separate or split the system.

Besides, from Figure 12 ($X \rightarrow Z$) and Figure 15 ($Z \rightarrow \dot{X}$), the state incidence matrix can be built differently than in Figure 11, by passing through the events as it is shown in Figure 17.

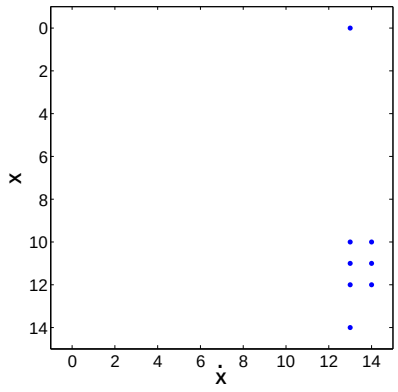

Figure 17. Incidence state matrix: derivatives of state variables $\dot{X}$ influenced by state variables X

Using this construction through the events Z, it appears that the state derivative $\dot{X}_{14}$ is also depending on X_{10}, X_{11} and X_{12}. Therefore, in order to determine correctly the relationships between the variables, it is important to use all the available system data, directly and by transitivity.

6.3 Incidence Event Matrix

The incidence event matrix can be built by transitivity. In fact, using the incidence matrix that define $Z \rightarrow D$ (see Figure 13) and conversely the matrix that define $D \rightarrow Z$, the incidence event matrix $Z \rightarrow Z$ can be deduced as it is shown in Figure 18.

As shown previously, it is hard to split the system based on the relationship between events Z and discrete variables D. However, with the incidence event matrix, it is possible to transform it into a block-diagonal form with three blocks using PaToH and to consider each block as a subsystems where all the related discontinuities belong to the same entity (see Figure19).

These blocks can be parallelized and we can hope the execution time to be reduced. In fact, the event detection, the event location and the restart of the solver increase the integration time as shown in Figure 20. In short, for the mono-cylinder integrated by the variable-step solver LSODAR, the average execution speed drops down to 4 times in case of events handling, and sometimes even up

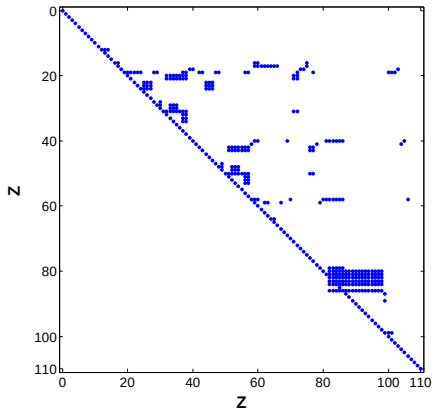

Figure 18. Incidence event matrix: events Z in columns influenced by events Z in rows

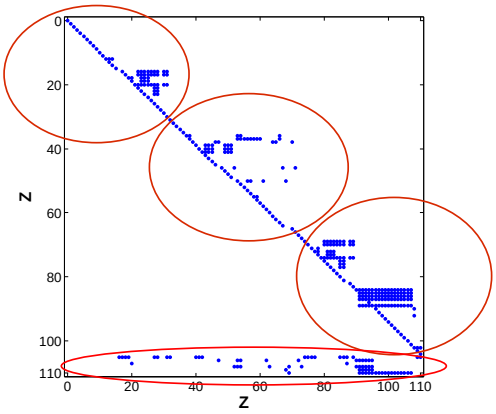

Figure 19. Block-diagonalized incidence event matrix: events Z in columns influenced by events Z in rows

to 60 times. This confirms the interest on both limiting the number of interrupts inside each block of the model due to the events and parallelizing the event location through the solver.

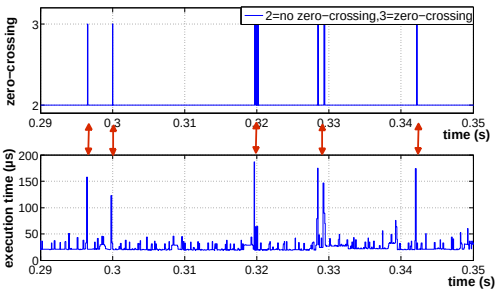

Figure 20. Effect of events handling on execution time

7. Summary and Future Directions

The various methods studied in the paper aim to contribute to speed-up the numerical integration of hybrid dynamical systems, eventually until reaching a real-time execution, while keeping the integration errors inside controlled bounds. Speed-ups can be achieved through an adequate partition of the original system where the interactions between the resulting sub-systems are minimized, so that they can be efficiently integrated in parallel.

Slackened synchronization, as analyzed theoretically in Section 4, allows for minimizing the number of integration interrupts and for using optimized integration parameters on each node (as illustrated experimentally in [2]) . However the corresponding induced delays between the subsystems must be limited to keep the integration errors under control.

The particular case study shows that it is not easy nor intuitive to know how to split a system, neither from a physical point of view nor from the relationship between the states and the events. In fact, the matrix between the coupled states and events is not sparse, so it is not possible to transform it into a block-diagonal form.

However, the incidence events matrix more likely seems to be sparse and its transformation to a block-diagonal form is feasible. Thus a relevant way to parallelize this particular system seems to perform it through the solver, leading to parallelize the steps corresponding to events handling which are costly for the numerical resolution (as already observed and plot in Figure 20).

Future works intend to practically evaluate the achievable speedups. This requires to extend the tool-chain of Figure 10, by developing a multi-thread runtime system able to take into account the parallelization choices presented in this paper. Then an engine with 4 cylinders will be studied to compare the split performed from a physical point of view in [2] to the combined use of the analytic approaches described in sections 4 and 5.

References

[1] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Philadelphia, PA, USA, 1st edition, 1998.

[2] A. Ben Khaled, M. Ben Gaïd, D. Simon, and G. Font. Multicore simulation of powertrains using weakly synchronized model partitioning. In *E-COSM'12 IFAC Workshop on Engine and Powertrain Control Simulation and Modeling*, Rueil-Malmaison, France, October 2012.

[3] Z. Benjelloun-Touimi, M. Ben Gaid, J. Bohbot, A. Dutoya, H. Hadj-Amor, P. Moulin, H. Saafi, and N. Pernet. From physical modeling to real-time simulation : Feedback on the use of modelica in the engine control development toolchain. In *8th Int. Modelica Conference*, Germany, March 2011. Linköping University Electronic Press, Linköpings universitet.

[4] F. Bergero, X. Floros, J. Fernández, E. Kofman, and F. E. Cellier. Simulating Modelica models with a stand-alone quantized state systems solver. In *9th Int. Modelica Conference*, Munich Germany, September 2012.

[5] G. Bernat, A. Burns, and A. Llamosi. Weakly hard real-time systems. *IEEE Trans. on Computers*, 50:308–321, April 2001.

[6] T. Blochwitz, T. Neidhold, M. Otter, M. Arnold, C. Bausch, M. Monteiro, C. Clauß, S. Wolf, H. Elmqvist, H. Olsson, A. Junghanns, J. Mauss, D. Neumerkel, and J.-V. Peetz. The functional mockup interface for tool independent exchange of simulation models. In *Proceedings of the 8th International Modelica Conference*. Linköping University Electronic Press, March 2011.

[7] G. D. Byrne and A. C. Hindmarsh. PVODE, an ODE solver for parallel computers. *International Journal of High Performance Computing Applications*, 13(4):354–365, Winter 1999.

[8] F. E. Cellier and E. Kofman. *Continuous System Simulation*. Springer, 1st edition, March 2006.

[9] C. Faure, M. Ben Gaïd, N. Pernet, M. Fremovici, G. Font, and G. Corde. Methods for real-time simulation of cyber-physical systems: application to automotive domain. In *IWCMC'11*, pages 1105–1110, 2011.

[10] M. C. Ferris and Jeffrey D. Horn. Partitioning mathematical programs for parallel solution. *Mathematical Programming*, 80:35–61, 1998.

[11] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica*. Wiley-IEEE Press, 2003.

[12] D. Guibert. *Analyse de méthodes de résolution parallèles d'EDO/EDA raides*. PhD thesis, Université Claude Bernard - Lyon I, Sep 2009.

[13] G. Karypis and V. Kumar. MeTiS : A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Technical report, Univ. of Minnesota, Dept. of Computer Science, 1998.

[14] E. Kofman and S. Junco. Quantized-State Systems: a DEVS approach for continuous system simulation. *Trans. of The Society for Modeling and Simulation International*, 18(3):123–132, September 2001.

[15] E. A. Lee. Computing foundations and practice for Cyber-Physical Systems: A preliminary report. Technical Report UCB/EECS-2007-72, Univ. of California, Berkeley, May 2007.

[16] M. Sjölund, R. Braun, P. Fritzson, and P. Krus. Towards efficient distributed simulation in Modelica using transmission line modeling. In *EOOLT*, pages 71–80, 2010.

[17] P. J. van der Houwen and B. P. Sommeijer. Parallel iteration of high-order Runge-Kutta methods with stepsize control. *J. Comput. Appl. Math.*, 29:111–127, January 1990.

[18] F. Zhang, M. Yeddanapudi, and P. Mosterman. Zero-crossing location and detection algorithms for hybrid system simulation. In *Proc. 17th IFAC World Congress*, pages 7967–7972, Seoul, South Korea, July 2008.

[19] Ü. V. Çatalyürek. *Hypergraph Models for Sparse Matrix Partitioning and Reordering*. PhD thesis, Computer Engineering and Information Science Bilkent University, November 1999.