Multicore simulation of powertrains using weakly synchronized model partitioning

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

Multicore simulation of powertrains using weakly synchronized model partitioning

Abir Ben Khaled [*] **Mongi Ben Gaid** [*] **Daniel Simon** [**] **Gregory Font** [*]

[*] *IFP Energies nouvelles, 1-4 avenue de Bois-Prau, 92852 Rueil-Malmaison, France*
(e-mails: {abir.ben-khaled, mongi.ben-gaid, gregory.font}@ifpen.fr).
[**] *INRIA, Inovalle Montbonnot, 38334 St Ismier Cedex, France*
(e-mail: daniel.simon@inria.fr).

Abstract: The need of quick innovation in the automotive domain made simulation necessary at early stages of the development cycle. Vehicles and powertrains are complex systems where different domains are involved. Representative phenomenological models of powertrains have been developed and have been used in the design phase under domain dedicated tools. However, their use for controls validation using Model-In-the-Loop (MIL) and Hardware-In-the-Loop (HIL) was prevented due to performance limitation of widely used single-solver/single-core simulation approaches.
This paper proposes a simulation approach for complex systems simulation, taking into account the fact that the computational power improvement in today processors is mainly driven by the augmentation of the number of cores per processor, rather than in cores frequency increase. It first focuses into numerical solvers characteristics, especially the time-step and order management, to identify their influence on the accuracy and the simulation speed. Then, a parallel simulation method is proposed in order to exploit more efficiently the parallelism provided by multi-core architectures, as well as involved complex systems nature (different dynamics and time scales), while avoiding disrupting simulation results. Finally, this method is validated on phenomenological engine test cases, and its effectiveness is illustrated.

Keywords:
Engine simulation, Powertrain simulation, Software-in-the-loop simulation, Multicore simulation, Numerical solvers

1. INTRODUCTION

Nowadays, it is strongly needed to rapidly adapt new engine concepts, design and related control strategies w.r.t. new regulations, in terms of fuel consumption and pollutant emissions reduction. Nevertheless, powertrains belong to complex systems category that require the design of both computational components (hierarchical controllers) and multidisciplinary physical models (mechanical, electrical, thermodynamic and chemical). Such highly complex systems composed of a large diversity of elements linked together by strong interactions need a systemic approach for the design and validation phases. In fact, the systemic approach to problems assumes that systems are seen as a whole, and that their parts must not just be seen individually (see de Rosnay [1997]). Moreover, the focus is only on the total inputs and total outputs of the system, without worrying on which part of the inputs goes to which subsystem (see Heylighen [1998]). Such an approach is concerned with total system performance even when a change in only one or some of its parts is considered. The reason is that there are some system's properties that can only be analyzed in an appropriate way from a holistic point of view.

With the systemic approach, all involved engineering fields start the design phase at the same time, and refines their models further when advanced. The coupling of different domain models does not require the knowledge of all the specialties on a very fine scale because the validation is achieved at a system level.

The classical engine has evolved towards a very complex system combining many hi-tech components with advanced control strategies. In this context, powertrain simulation tools have been shown to provide an indisputable support during all stages of engine developments. They can be used for various targets such as system understanding, design investigation, non-measurable value access or virtual bench for control and calibration. However, these tools require sophisticated models to be efficient, especially in the combustion chamber where combustion and pollutant formation processes take place. Moreover, these tools usually use a single solver for the resolution of the global systems simulation, and does not allow the exploitation of the available computational power of multi-core processors.

Today, there is an increasing need of a better control of combustion in order to reduce the consumption and the pollutant emissions. Therefore, the goal is to keep the same combustion model for all the applications (including Model-In-the Loop and Hardware-In-the-Loop) in order to have both accurate modeling and low CPU time. For this purpose, efficient methods of complex systems (or systems of systems) simulation are needed for avoiding model reduction for MIL and HIL applications.

The aim of this paper is to propose an approach for complex systems simulation, showing the possibility to use variable-step solvers, and investigating model decomposition approaches for multi-core co-simulation. First, characteristics of numerical solvers are presented to show their influence on both accuracy and simulation speed. After that, multi-core simulation is studied to show the effect of model decomposition on the numerical integration. Finally, a case study concerning a phenomenological engine model is presented and test results are analyzed from the execution time and precision points of view.

2. OVERVIEW OF NUMERICAL SOLVERS

A numerical solver must be chosen in order to solve the model made of a system of equations (Ordinary Differential Equations ODEs). Therefore, it is necessary to define the essential criteria and to satisfy them by determining the parameters of numerical integrators.

2.1 Criteria of numerical schemes

To measure the performances of an integrator and to compare it with others, some criteria are employed:

Consistency The consistency is a property of the discretization. The discretized ODE is consistent with respect to the real ODE if it tends to it when the time-step dt tends to zero. The difference between the discretized equation and the true equation is called the truncation error (or theoretical error, or global consistency error) (see Butcher [2008]).

Stability The stability represents the amplification/ attenuation of the errors during computing (round-off, truncation, method), unconditional or under conditions (discretization step,..). It characterizes the propagation of an initial perturbation during an entire numerical integration.
For a given integrator, it is important to quantify its stability. So, in order to have stability, it is necessary that all the eigenvalues of the matrix of amplification A are less than one in absolute value. The matrix of amplification A being as follows: $X_{n+1} = A * X_n$ with X_n is the state vector at $t = n.dt$ and X_{n+1} at $t = (n+1).dt$.

Accuracy (Convergence) The convergence is a property of the numerical solution. The numerical solution converges to the analytical solution if it tends to it at any point of time when dt tends to zero (i.e. all local discretization errors tends to zero).

The consistency and the stability are necessary and sufficient for the convergence, (see Lax and Richtmyer [1956]). In other words, if an ODE is discretized in a consistently way and if the solution of this ODE is stable, then it is convergent.

Speed For a given model and computing resource, the fastness of a solver depends on the total amount of computations required to resolve the system of equations. To achieve real-time simulations, the challenge is in particular to find a satisfactory trade-off between the integration speed and precision which usually leads to conflicting constraints.

2.2 Parameters of numerical schemes

To satisfy the chosen criteria some parameters of the numerical integrators are employed:

Order This reflects the accuracy of the numerical solution. A method has an order n, when it neglects in the Taylor series all terms where the order is higher than n (e.g. Euler is 1st order). The local truncation error is proportional to $O(dt^{n+1})$ and the global truncation error is proportional to $O(dt^n)$.

Explicit/Implicit For explicit scheme (e.g. explicit Euler, Adam Bashfort), X_{n+1} is computed directly from past values $X_n, X_{n-1}, ...$ but for implicit scheme (e.g. Cranck Nicholson, Adam Moulton), the computation of X_{n+1} needs the resolution of an additional equation (often non-linear).

Compared with explicit schemes, implicit schemes are often less accurate during the initial steps. They are also more complex to implement and require more computations at each time-step, because they need the resolution of a non-linear system based on Newton iterations at each integration step. Consequently, the cost of each integration step is considerably more expensive than in the case of an explicit algorithm. However, implicit algorithms are far more stable and effective when integrating stiff systems (where the model merges subsystems with very different decay rates and time constants), whereas explicit schemes need to use tiny time-steps to ensure stability, or even totally fail due to numerical instability.

Fixed step/Variable (adaptive) step In general, the time-step dt must be much smaller or even negligible compared to the system's dynamics in order to have stability. An efficiency problem arises when the system's temporal behavior changes over the simulation horizon, for example when fast transients are mixed with slower state evolutions. When using fixed time-step methods, the step size must be chosen tiny enough to comply with fast dynamics, thus wasting CPU cycles when integrating the model in its slow behaviors.. However in that case the number of integration steps can be known (according to the method's order), so that the execution time is predictable. For adaptive methods (e.g. RK45 Fehlberg), the time-step size is driven by the integration error. A feedback loop adapts the step size according to the integration error estimate. Iterations are done until a predefined bound on the error is achieved, thus leading to overheads and unpredictable integration time.

In other words, the integration step management give a choice between time driven integration (fixed-steps) and error driven integration (variable-steps).

One-step/Multi-step One-step methods (e.g. Runge-Kutta) only use the current values of X_n and $f = dX_n/dt$, they depend on evaluations of the differential equation at well-chosen locations within the current integration interval. Multi-step methods use several past values of X_n (e.g. backward difference formula BDF) and f (e.g. Adams) to achieve a higher order of accuracy. BDF is the most effective multi-step method for stiff systems. Newton iteration is used to solve the nonlinear system at each time-step, which represents almost the total cost in solution computation. Adams methods are the best known multi-step methods for solving general non stiff systems, where the non-linear system is solved by a simple Functional Iteration.

2.3 Classification of numerical integrators

There is a balance between the computation time, required accuracy and stability. For fixed-step solver applied to a simple model, its behavior is summarized in table 1.

Table 1. Preliminary classification of numerical schemes

Solver	Speed	Stability	Accuracy
$\downarrow$ order, explicit	fast	hardly stable	accurate
$\downarrow$ order, implicit	fairly fast	very stable	accurate
$\uparrow$ order, explicit	slow	stable	accurate
$\uparrow$ order, implicit	very slow	very stable	very accurate

For complex model behaviors, as hybrid systems, systems with a huge number of unknowns or stiff systems, there is no general rule for choosing a particular method or solver. An important point, especially when dealing with hardware-in-the-loop (HIL) real-time simulation where the execution time is constrained, is that there is a trade-off between simulation speed and accuracy for every chosen method.

The LSODAR integration package (see Hindmarsh and Petzold [1995]) is based both on the BDFs and Adams methods and automatically switch between them when the system alternates between stiff and non-stiff behaviors. It has also a root-finding capability in order to detect events such as zero-crossing of state variables.

3. PARALLELIZATION APPROACHES FOR HYBRID ODES RESOLUTION

When complex hybrid ODEs have to be solved, simulation duration becomes of primary concern. To allow a speedup of this simulation duration and to avoid model reduction, parallelization of the simulation resolution is of interest.

3.1 Introduction

Parallel computing is employed for solving large problems that could be partitioned into many independent small parts in order to be solved by multiple processing elements in a simultaneous way. There are different types of parallelism: bit-level, instruction level, data and task or thread parallelism.

Bit-level parallelism is directly related to the processor word size. In fact, when variables sizes are greater than the length of the word, the processor have to execute an operation in at least two instructions. So increasing the word size reduces the number of instructions.

Instruction-level parallelism is directly related to the processor pipeline size. In fact, the number of stages in the pipeline represents the potential parallelism for the instructions. In order to have an efficient parallelism, there have to be many independent instructions that could be re-ordered and grouped together into a pipeline. Recent processor have super-scalar capabilities allowing them to be able to execute several instructions in parallel, using algorithms like Tomasulo algorithm.

Data parallelism is directly related to program loops. It is possible only when there is no dependencies in the iteration loop. The iterations could be then divided into the number of available processors and executed in parallel without disturbing the data.

Task parallelism consists in distributing threads (or process) across multiple core or processors. It allows different calculations on the same or different sets of data, unlike the data parallelism which only allows the same calculation on them.

Different approaches have been proposed for the parallelization of ODEs and hybrid ODEs:

3.2 Parallelization across the method

The main approach to obtain a parallel numerical scheme for ODE systems consists of parallelizing "across the method". This 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 van der Houwen and Sommeijer [1990] and Guibert [2009]) and parallelizing operations on vectors for ODEs resolution by separating them into modules (see PVODE in Byrne and Hindmarsh [1999] implemented using MPI (Message-Passing Interface) technology).

3.3 Parallelization across the steps

Another approach, which is the time decomposition method, originally introduced by researchers from the multi-grid field (Horton and Vandewalle [1993]) and applied to solve Partial Differential Equations (PDEs). Moreover, two methods was introduced in this category: the Parareal scheme proposed in Lions et al. [2001] and PITA algorithm described in Farhat and Chandesris [2003], where both of them were derived from the multiple shooting method (see Deuflhard [2004]). They follow the approach of splitting the time domain in sub-domains by considering two levels of time grids. A first parallel computation of a predicted solution is performed with a fine time grid. After that, at each end of time of sub-domains, the solution makes a jump with the previous initial boundary value (IBV) of the next time sub-domain. A correction of the IBV for the next fine grid is then computed on the coarse time grid. In Farhat and Chandesris [2003], it shows that the method converges at least in a finite number of iterations. Nevertheless, this approach seems to have difficulties for stiff non linear problems. In Guibert [2009], adaptivity in the tolerance of the time slice integrator and the number of sub-domains was studied and some improvements has been shown towards stiff ODEs. However, for very stiff problems, difficulties still appears. So another parallel solver has been proposed for stiff ODEs based on Richardson Extrapolation.

3.4 Parallelization across the model

Numerically integrating PDEs in parallel is facilitated by the need to solve across their spatial dimensions, which naturally leads to data parallelism. However, this is not the case for ODEs and DAEs (Differential Algebraic Equations), when both models and solvers exhibit a strong sequential nature.

Decoupling this apparently sequential computation is an important issue for distributed simulation, because the way of decoupling a system could significantly affect the simulation results. Some important methods exploiting the parallelization across the model include the relaxation algorithm and the transmission-line modeling method.

The Waveform Relaxation (WR) method was introduced in Lelarasmee et al. [1982]. It is an iterative process originating from Picard theorem, which makes possible to solve simultaneously in parallel coupled subsystems over successive time windows. It consists in, first, decomposing the system into several subsystems, then, solving each of them with an initial condition. In each iteration, every subsystem is solved separately in order to find its waveform (i.e. solution), considering all the waveforms of the other subsystems constant. The iteration stops when the solutions converge. In order to achieve a reasonable convergence rate, the system should be partitioned at specific locations where the coupling is weak. This is a trade-off for the designer because if there are too many tightly coupled elements in a same subsystem, the advantage of using the relaxation algorithm is weakened. In Hui and Christopoulos [1990], three types of coupling methods were compared to show how they affect the convergence of the solution.

The Transmission Line Modeling (TLM) technique (see Hui and Christopoulos [1990]) is a discrete modeling method that provides a general approach for decoupling systems in distributed simulation. It represents the physical process by a transmission-line graph which is solvable by a computer. According to this method, the decoupling point should be chosen where variables change slowly and the time-step of the solver should be relatively small. Consequently, the decoupled subsystems are seen as they were connected by constant variables and the error due to time delays can be significantly decreased. Other advantages of this technique are the ability to increase the efficiency and the accuracy of the solution, in fact the discrete model is derived with TLM directly from the physical system (elimination of the discretization error).

Other work around the parallelization across the model have been conducted in Faure et al. [2011] in the context of Hardware-In-The-Loop. Several alternative methods were proposed in order to perform real-time simulation of complex physical models. However, the study was focused only on fixed-step solver without treating the case of variable-step solver.

4. PROPOSED MULTI-CORE SIMULATION APPROACH

The case study of interest presents highly coupled components, with strong interactions: the model graph execution does not present any evident parallel branches, because the global model showed interdependent interactions between its components. The parallelization across the model approach was selected to tackle the problem of the multi-core simulation of this case study. This approach requires decoupling some model parts that seem a priori coupled from a model execution graph point view. The decoupling approach is illustrated in the following.

4.1 Model decomposition approach

The model is partitioned in order to balance computations made by each partition. The partitioning is chosen so that the time constants that are closest are grouped into homogeneous subsystems. This allows isolating stiff parts.

The second goal is to decrease the number of discontinuities affecting each subsystem at runtime. When using a variable-step solver like LSODAR, after the initial phase the integration step can be made very large (according to the desired accuracy) and considerable speedups can be obtained compared with a fixed-step solver.

However, events switches in the model may occur at some particular point of the system's state space, for example at the ignition time of one cylinder in an engine. To keep the integration accuracy, the integration process must be accurately stopped and restarted at this point with a new set of parameters.

For fixed-step solver, the step size must be chosen small enough so that these particular events can be captured within the width of one step. For variable-step solver, these events are captured by a root-finding process [1]. The drawback is that at each interrupt the variable-step integrator must re-initialize and restart with small step size, thus inducing a penalizing overhead.

Therefore, it is expected that minimizing the interruptions asynchronously occurring between subsystems would leave them integrating at high speed for longer periods of time. For example, if every cylinder of an engine runs in a separate subsystem, it would not be disturbed by the ignition events coming from the other cylinders.

[1] The root-finding process is potentially unbounded, but its execution time can be guarded by a well chosen threshold

These two ideas, decreasing stiffness and minimizing interrupts between subsystems, are assumed to speedup the integration process even on single core processors through a well suited partition of the global system, i.e. running a pseudo-parallel execution of the model integration in threads synchronized w.r.t. irreducible variable dependencies between subsystems.

Finally, model partitioning can be further actually run in a parallel execution on a multi-core architecture, taking advantage of increased computing power to speedup the whole process. In that case, the subsystems are periodically synchronized to exchange data between them; this period is denoted communication time-step. Each subsystem can be integrated by its own solver, which can be chosen and tuned according the characteristics and integration needs of the set of equations.

4.2 Model decomposition effect on numerical integration

Partitioning a continuous model made of a set of ODEs raises the question of dependencies and synchronization between the subsystems. Remind that the goal is to provide an end-to-end real-time simulation with respect to a predefined precision.

The subsystems and associated numerical integrators read inputs and send outputs from/to the others at each communication step. A strict respect of dependencies between subsystems leads for waiting periods and idle processor time, therefore making decreasing the efficiency of the overall model integration. To increase the parallelism of the concurrent execution of the partitioned model, it is proposed to slacken the time dependencies constraints between the subsystems. It is argued later on that the induced integration imprecision can be kept small enough and compliant with the specification thanks to a more efficient utilization of the available computing power at hand. The idea is enlighten through the following example.

Let the continuous system be

$$\begin{cases} \dot{X}(t) &= F(X(t), U(t)) \\ Y(t) &= G(X(t), U(t)) \end{cases}$$

with U the input, X the state vector, and Y the output.

Assume that the system is split in 2 subsystems (see figure 1) as follows:

$$\dot{X} = \begin{pmatrix} \dot{X}_1 \\ \dot{X}_2 \end{pmatrix}, \; Y = Y_2, \; U_1 = U \text{ and } U_2 = Y_1$$

with:

$$\begin{cases} \dot{X}_i(t) &= F_i(X_i(t), U_i(t)), \quad \text{for } i = 1,2 \\ Y_i(t) &= G_i(X_i(t), U_i(t)), \quad \text{for } i = 1,2 \end{cases}$$

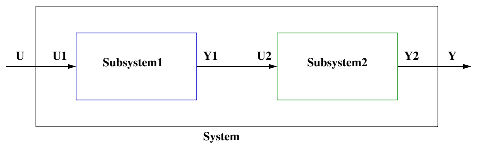


Fig. 1. Example of system decomposition

In the following, Euler method will be taken as an example to solve the system on a dual-core, the integration step will be equal to the communication step.

Using the Euler integration, each solution ($X_1(t+h)$ and $X_2(t+h)$) is resolved in a single task, with a computation time C:

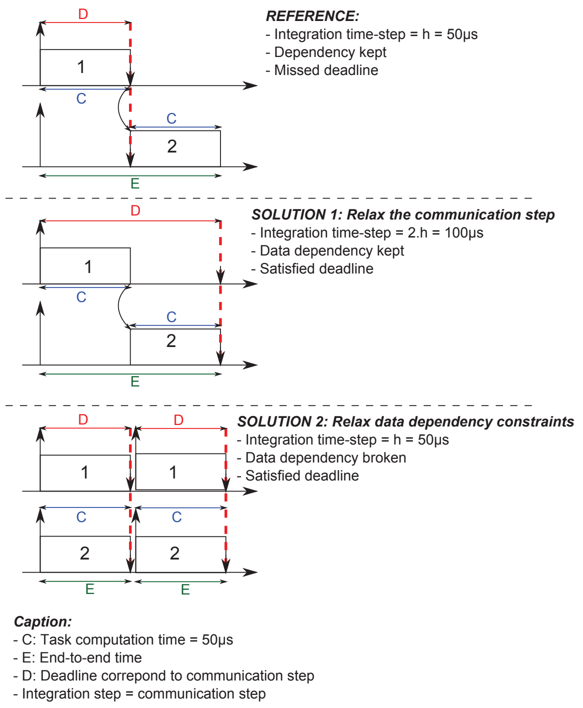
$X_i(t+h) \approx X_i(t) + h.F_i(X_1(t), U_i(t))$ for $i = 1, 2$.

Here h represents the reference time-step, as well as task execution period. It is chosen small enough in order to consider the solutions X_1 and X_2 satisfying regarding accuracy. However, these solutions don't meet the real-time constraints. In fact, the end-to-end time E is longer than the deadline D (in this example it is twice greater), where D represents the real-time constraints imposed by the outside environment and corresponds to the communication step (see figure 2).

In order to have coherence between the simulated time and the real time, two solutions are proposed. Note that the execution of a split system on a multi-core machine may affect the result of the numerical scheme. Through the following cases, it will be shown that keeping the dependencies (thus imposing $U_2(t) = Y_1(t)$) between the models or breaking them has an impact on the equations resolution.

- 1st solution: Relax the communication step

In order to comply with real time constraints, the deadline needs to be relaxed and taken twice greater. Which means that the integration time-step is equal to $2 * h$. Still, dependencies are kept, so subsystem 2 depends on subsystem 1 (see figure 2).



D
REFERENCE:
- Integration time-step = h = 50µs
- Dependency kept
- Missed deadline
1
C
C
2
E
D
SOLUTION 1: Relax the communication step
- Integration time-step = 2.h = 100µs
- Data dependency kept
- Satisfied deadline
1
C
C
2
E
D
D
SOLUTION 2: Relax data dependency constraints
- Integration time-step = h = 50µs
- Data dependency broken
- Satisfied deadline
1
1
C
C
2
2
E
E
Caption:
- C: Task computation time = 50µs
- E: End-to-end time
- D: Deadline correpond to communication step
- Integration step = communication step

Fig. 2. Integration behavior of dependent subsystems

Here $U_2(t) = Y_1(t)$ because the subsystem 2 wait the subsystem 1 to finish, so it gets the current output.

Let us denote for this case, X_i° and Y_i° instead of X_i and Y_i, for $i = 1, 2$, when using time-step larger than the reference h.

Using Euler with a time-step $2 * h$ leads to :

$$\begin{cases} X_1^\circ(t+2h) &= X_1^\circ(t) + 2h.F_1(X_1^\circ(t), U_1(t)) \\ Y_1^\circ(t+2h) &= G_1(X_1^\circ(t+2h), U_1(t+2h)) \end{cases}$$

$$\begin{cases} X_2^\circ(t+2h) &= X_2^\circ(t) + 2h.F_2(X_2^\circ(t), Y_1^\circ(t)) \\ Y_2^\circ(t+2h) &= G_2(X_2^\circ(t+2h), Y_1^\circ(t+2h)) \end{cases}$$

at $t+2h$:

$$\begin{cases} Er^\circ(Y_1) &= |Y_1(t+2h) - Y_1^\circ(t+2h)| \\ Er^\circ(Y_2) &= |Y_2(t+2h) - Y_2^\circ(t+2h)| \end{cases}$$

- 2nd solution: Relax data dependency

In order to comply with real time constraints, the integration time-step is kept equal to h whereas the data dependency is broken in order to have an end-to-end time E equal to the computation time C instead of $2C$ (see figure 2). Here the CPU is used efficiently at 100% load.

Here $U_2(t) = Y_1(t-h)$ with $U_2(0) = 0$ because subsystem 2 does not wait for subsystem 1 to finish, therefore it gets its previous output.

Let us denote for this case, X_i^* and Y_i^* instead of X_i and Y_i, for $i = 1, 2$, because the dependencies are ignored.

Using Euler with a time-step h:

$$\begin{cases} X_1^*(t+h) &= X_1^*(t) + h.F_1(X_1^*(t), U_1(t)) \\ Y_1^*(t+h) &= G_1(X_1^*(h), U_1(h)) \end{cases}$$

Here the output $Y_1^* = Y_1$, so the error is equal to zero.

$$\begin{cases} X_2^*(t+h) &= X_2^*(t) + h.F_2(X_2^*(t), Y_1^*(t-h)) \\ Y_2^*(t+h) &= G_2(X_2^*(t+h), Y_1^*(t)) \end{cases}$$

at $t+2h$: $Er^*(Y_2) = |Y_2(t+2h) - Y_2^*(t+2h)|$

In order to illustrate the previously computed error, the following example will show the evolution of the integration error for both cases, where Subsystem 1 and Subsystem 2 are assumed to be first order systems with transfer function $\frac{1}{1+\tau.p}$ with $\tau = 0.5\ s$.

Figure 3 shows that larger is the time-step, larger is the relative error.

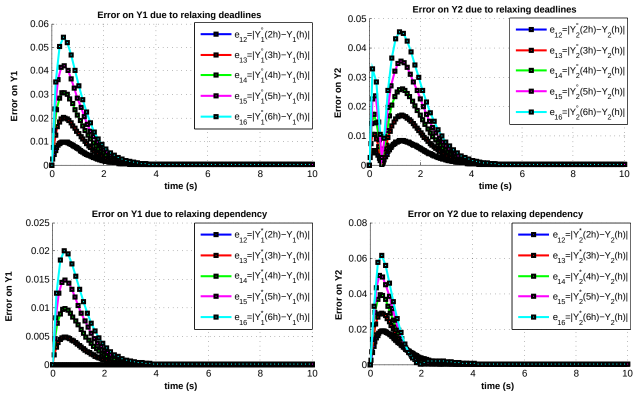


Fig. 3. Effect of time-step variation on accuracy

In order to compare the two solutions, the accumulated integration errors are plotted for different time-steps (see Figure 4).

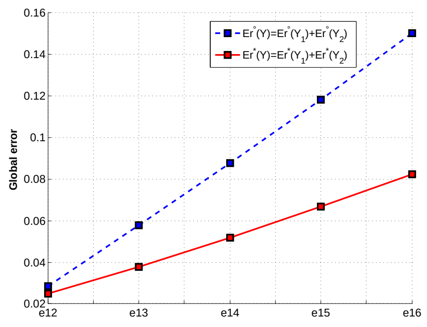


Fig. 4. Total error on Y1 and Y2 over time-step

By adding the outputs errors, breaking the data dependencies could be the best solution in term of shorten computation time with keeping an appropriate accuracy.

However, if the interest is on the maximum error for each subsystem, the choice between the two cases will be harder to make. For example, for the time-step equal to $2h$, $Er^{\circ}(Y_1) > Er^{*}(Y_1)$ and $Er^{\circ}(Y_2) < Er^{*}(Y_2)$.

5. CASE STUDY

5.1 Engine description

In this study, a Spark Ignition (SI) RENAULT F4RT engine has been modeled. It is a four-cylinder in line Port Fuel Injector (PFI) engine in which the engine displacement is 2 L. The combustion is homogeneous. The air path consists in a turbocharger with a mono-scroll turbine controlled by a waste-gate, an intake throttle and a downstream-compressor heat exchanger. To finish, this engine is equipped with two Variable Valve Timing (VVT) devices, for intake and exhaust valves. The maximum power is about 136 kW at 5000 rpm.

5.2 Engine modeling and simulation

The F4RT engine model was developed using ModEngine library (Benjelloun-Touimi et al. [2011]). ModEngine is a Modelica (Fritzson [2003]) library that allows the modeling of a complete engine with diesel and gasoline combustion models. Requirements for the ModEngine library were derived from the existing IFP-Engine AMESim [2] library. ModEngine contains more than 250 sub models. It has been developed to allow the simulation of a complete virtual engine using a characteristic time-scale of the order of the crankshaft angle. A variety of elements are available to build representative models for engine components, such as turbocharger, wastegate, gasoline or Diesel injectors, valve, air path, EGR loop etc... ModEngine is currently functional in Dymola tool [3] .

The previously presented parallel simulation methods were implemented into xMOD model integration and virtual experimentation tool (Ben Gaid et al. [2010]). F4RT model (and split parts) were imported into xMOD using the functional mock-up interface (FMI) (Blochwitz et al. [2011]) export features of Dymola. Specifically, the FMI standard describes the software interface of a hybrid ODE system.

[2] http://www.lmsintl.com/imagine-amesim-1-d-multi-domain-system-simulation
[3] http://www.3ds.com/products/catia/portfolio/dymola

5.3 Combustion model description

Two different types of combustion models have been used in this study.

- The first one is the phenomenological CFM-1D model developed by IFPEN (Richard, S. et al. [2009]) and based on the reduction of the 3D ECFM model (Colin, O. et al. [2003]). In this model, the rate of fuel consumption depends on the flame surface, computed thanks to the laminar flame speed and the turbulent kinetic energy. Only one parameter related to turbulent kinetic energy is tuned for combustion calibration. The other ones remain constant for the whole operating conditions. The CFM-1D model is the typical modeling level able to combine a good representation of physical phenomena with reasonable CPU performances. Thanks to these characteristics, this model can be embedded in a full engine simulator and used for architecture design or control strategy development issues (Richard et al. [2011]).
- The second one is a semi-physical model, based on the Wiebe model for combustion heat released (Wiebe [1970]), and presenting much less complexity. It is based on a mix of physical approaches and identifications or learning processes applied on the results of an experimental or/and numerical combustion campaign performed with a more complex model. The main advantage of this model is to take into account the behavior of the engine with a crank angle degree timescale, which is not the case of look-up table models.

Wiebe-based engine model has 87 continuous states and 420 event indicators (of discontinuities), whereas CFM-based model has 118 state variables and 398 event indicators.

5.4 Decomposition of the case study

The partitioning of the engine model is performed by separating the combustion cycle in the four-cylinder from everything else, called airpath, then by isolating the combustion cycle from each cylinder.

This kind of splitting is interesting for variable time-step solver because it relaxes event constraints i.e. decreases the number of events. In fact, the combustion phase presents many events and it is executed from one cylinder to another in a sequential way. Then, at the end of the last cylinder combustion, the cycle is repeated relentlessly. By splitting the models, the solver can treat locally the events during combustion cycle in a single cylinder and then relax time-step until the next cycle.

5.5 Test results

In the following tests, simulations of F4RT engine are done in xMOD for both Wiebe and CFM combustion models. As a first approach, the idea is to compare the variable-step solver LSODAR against the fixed-step solver RK4 with a small integration step size ($50\mu s$), considered as a reference by model developers. The validation will be performed using the quantities of interest as intake and exhaust manifold pressures, air-fuel equivalence ratio (AFR) and torque.

Accuracy with variable-step solver
Figure 5 shows the intake manifold pressure and the torque during 2 engine cycles (using a Wiebe model with an engine speed equal to $2500rpm$). These outputs are computed using both LSODAR with a communication time-step equal to $500\mu s$

and a tolerance equal to 10^{-5} and RK4 with a time-step equal to $50\mu s$. The accuracy is ensured, in fact the error between the outputs of manifold pressure is less than 0.3% and for the torque is less than 0.5%

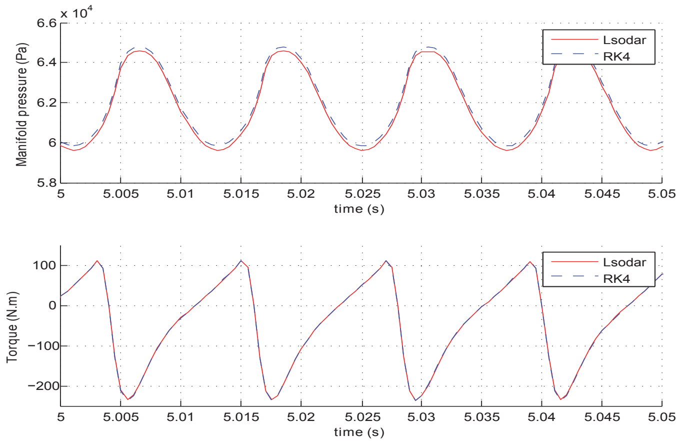


Fig. 5. Several outputs using RK4 and LSODAR solvers

With variable-step solver, the bounding of the error due to the integration is ensured. However, at the same time, the execution time is 4 time longer with a tolerance equal to 10^{-4} and 6 time longer with a tolerance equal to 10^{-5}. This is due to the presence of a large number of discontinuities that decreases the speed advantage of variable-step solvers.

Events are related usually to the evolution of a subset of the state vector, so the partitioning of the engine model will be performed from a physical point of view in order to minimize the number of integration interrupts.

Model splitting effect on execution time with single-core
The first step is to compare the execution time between the original model and the split model but executed on a single-core, in order to only see the effect of events relaxation on the speedup of LSODAR solver without the effect of the parallelization.

- Result 1: Number of discontinuities

The partitioning of the model involved the decrease of the number of the discontinuities seen by the solver. In fact, tests during 0.3 s show that the unpartitioned model presents 851 events whereas the split model presents on average 203 events per cylinder and 119 for the airpath. Figure 6 summarizes that during 2 engine cycles.

- Result 2: Integration step size

The impact of events reduction per subsystems involves the decrease of the number of integration interrupts, so the increase of the time-step size as shown in figure 7. For the global model, the maximum step size is around 422 μs and the mean value is about 148 μs whereas for the split model, the step size reaches the maximum allowed one 500 μs and the mean value is around 215 μs for the cylinders and 229 μs for the airpath.

- Result 3: Execution time

Results 1 and 2 entail a speedup of the execution time, about 1.98 without the use of multi-core parallelization.

Model splitting effect on execution time with multi-core
In this section, the interest is on the parallelization of the model using a multi-core PC. Tests are done using RK4 solver on a CFM model. The engine model is executed as shown in figure 8 using at first 2 cores then 4 cores. After that, the inter-subsystem dependencies are ignored and the model is executed on 5 cores.

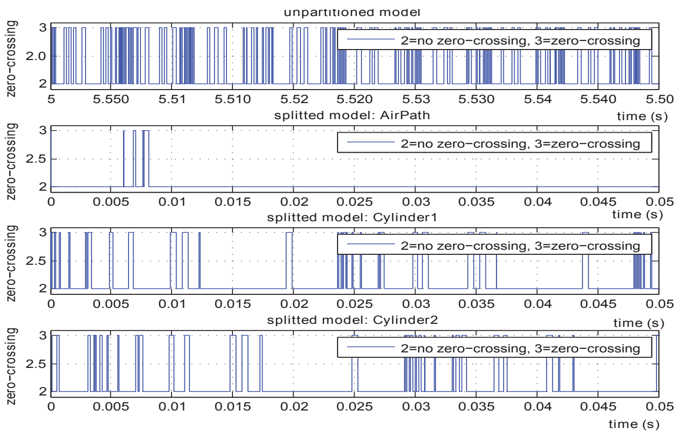


Fig. 6. Number of discontinuities per model

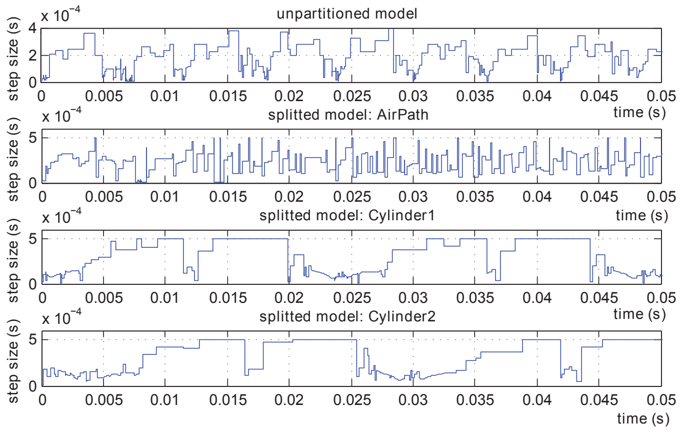


Fig. 7. Integration step behavior per model

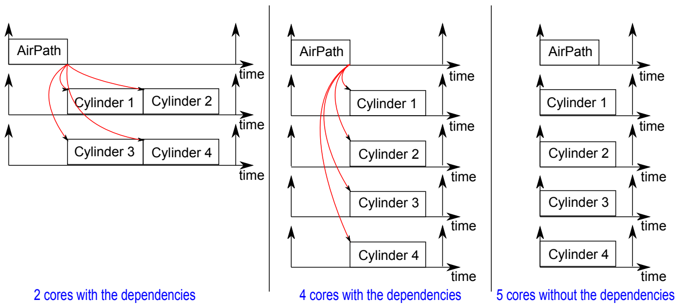

AirPath
Cylinder 1
Cylinder 2
Cylinder 3
Cylinder 4
time
2 cores with the dependencies
4 cores with the dependencies
5 cores without the dependencies

Fig. 8. Distribution of the engine model

Test results show that the speedup of execution time is about 1.77 when using 2 cores, then it is increased to 3.15 when using 4 cores. For the last case, the speedup is about 3.9. The question then is “what is the best regarding execution time and accuracy, ignoring these dependencies and running the model at 50 μs, or keeping them and running it with 100 μs”.

Table 2 shows that ignoring the dependencies between the airpath and the four cylinders and using RK4 with the time-step 50 μs presents less error in major outputs than keeping the dependencies but integrating with a time-step equal to 100 μs.

Regarding the execution time, the second case is faster than the first one, the speedup is around 2.06. We can conclude then that executing the F4RT engine with the CFM model under

Table 2. Error on several outputs

	Case 1[4] :error(%)	Case 2[5] :error(%)
Air-fuel equivalence rate	1.02	0.53
Intake Pressure	0.47	0.42
Compressor Speed	0.65	0.50
Exhaust Pressure	0.93	0.94
Torque	7.32	0.55

a multi-core machine is better in term of execution time and accuracy, when the dependencies between the airpath and the cylinders are ignored with a time-step equal to 50 μs, than when the dependencies are respected with a time-step equal to 100 μs.

6. CONCLUSION

In this paper, multicore simulation for complex systems has been studied with a focus on simulation duration speedup. The methodology of parallelization across the model has been selected for such problem where strong interactions between the model components are observed. The current study showed that decoupling the model parts by relaxing their data dependencies is promising in term of simulation speed (by increasing the parallelism) and results accuracy. Besides, tests results on engine model showed that, with the model partitioning, it is possible to use efficiently variable-step solvers thanks to the decrease of the number of discontinuities, so the number of integration interrupts, in each subsystem

Further work will investigate in the combination of the use of variable-step solvers in split model with the use of multicore architecture for parallel computing, in order to improve the simulation speedup while keeping results accuracy under control.

REFERENCES

M. Ben Gaid, G. Corde, A. Chasse, B. Lety, R. De La Rubia, and M. Ould Abdellahi. Heterogeneous model integration and virtual experimentation using xMOD: Application to hybrid powertrain design and validation. In *7th EUROSIM Congress on Modeling and Simulation*, Prague, Czech Republic, Sep. 2010.

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 International Modelica Conference*, Germany, March 2011.

T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauss, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. The functional mockup interface for tool independent exchange of simulation models. In *8th International Modelica Conference*, Germany, March 2011.

J C Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd, 2008.

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.

Colin, O., Benkenida, A., and Angelberger, C. A 3D modeling of mixing, ignition and combustion phenomena in highly stratified gasoline engines. *Oil & Gas Science and Technology*, 58:47–62, 2003.

J. de Rosnay. Analytic vs. systemic approaches. Principia Cybernetica Web, 1997. URL `http://pespmc1.vub.ac.be/ANALSYST.html`.

P. Deuflhard. *Newton Methods for Nonlinear Problems*, volume 35 of *Springer Series in Computational Mathematics*. Springer, 2004.

C. Farhat and M. Chandesris. Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications. *International Journal for Numerical Methods in Engineering*, 58(9):1397–1434, 2003.

C. Faure, M. Ben Gaid, 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.

P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica*. Wiley-IEEE Computer Society Pr, 2003. ISBN 0471471631.

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

F. Heylighen. Basic concepts of the systems approach. Principia Cybernetica Web, 1998. URL `http://pespmc1.vub.ac.be/sysappr.html`.

A. C. Hindmarsh and L. R. Petzold. Algorithms and software for ordinary differential equations and differential-algebraic equations, part II: higher-order methods and software packages. *Comput. Phys.*, 9:148–155, March 1995. ISSN 0894-1866.

G. Horton and S. Vandewalle. A space-time multigrid method for parabolic pdes. Technical report, Universitaet Erlangen, 1993.

S. Y. R. Hui and C. Christopoulos. Numerical simulation of power circuits using transmission-line modelling. *IEE proceedings. Part A. Physical science, Measurements and instrumentation, Management and education, Reviews*, 137 (6):379–384, 1990.

P. D. Lax and R. D. Richtmyer. Survey of the stability of linear finite difference equations. *Communications on Pure and Applied Mathematics*, 9(2):267–293, 1956.

E. Lelarasmee, A.E. Ruehli, and A.L. Sangiovanni-Vincentelli. The waveform relaxation method for time-domain analysis of large scale integrated circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 1(3):131–145, Jul 1982.

J. Lions, Y. Maday, and G. Turinici. Résolution d'EDP par un schéma en temps "pararèel". *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 332(7):661–668, April 2001.

S. Richard, G. Font, F. Le Berr, O. Grasset, and M. Fremovici. On the use of system simulation to explore the potential of innovative combustion systems: Methodology and application to highly downsized SI engines running with ethanol-gasoline blends. *JSAE Paper*, 2011-04, 2011.

Richard, S., Bougrine, S., Font, G., Lafossas, F.-A., and Le Berr, F. On the reduction of a 3D CFD combustion model to build a physical 0D model for simulating heat release, knock and pollutants in SI engines. *Oil & Gas Science and Technology*, 64:223–242, 2009.

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.

[4] with dependencies and time-step=100 μs

[5] without dependencies and time-step=50 μs

I.I. Wiebe. *Brennverlauf und Kreisprozess von Verbrennungsmotoren*. VEB Verlag Technik, Berlin, 1970.