Towards a Weakly-Hard Approach for Real-Time Simulation

Abir Ben Khaled Mongi Ben Gaid
IFP Energies nouvelles,
1-4 avenue de Bois-Préau,
92852 Rueil-Malmaison, France
abir.ben-khaled@ifpen.fr mongi.ben-gaid@ifpen.fr

Daniel Simon
INRIA
Inovallée Montbonnot
38334 St Ismier Cedex, France
daniel.simon@inria.fr

Abstract

New regulations for vehicles are increasingly demanding on fuel consumption and pollutant emissions reduction. This requires to design new engine concepts and related control strategies. Control design, validation and calibration on test beds is costly and time consuming. To reduce development costs and time-to-market, Electronic Control Units (ECUs) have to be validated at an early stage using a real-time simulation platform based on continuous engine models.

To allow real-time simulation of high fidelity engine models, different techniques have to be applied in order to fulfill the real-time constraints. Real-time simulation involves trade-offs between several aspects, such as real-time constraints, models computational complexity and integration accuracy. This paper mainly focus on numerical integration related aspects, by introducing its different methods such as numerical solvers and partitioning/scheduling, and then considering the trade-off between accuracy and simulation speed.

1. Introduction

Designing complex systems like Cyber-Physical Systems which include both physical and computational models is time consuming and implies the knowledge and cooperation of different disciplines [7]. In order to reduce design, development and validation phases, a global simulation is needed at an early stage. The main purpose of the numerical simulation is, when an analytical solution cannot be derived, to approximate its behavior as faithfully as possible. In other words, bounding and minimizing the simulation errors is the important aim during numerical simulation.

Besides, in a hardware-in-the-loop (HIL) architecture, real components (e.g. an ECU) are linked with simulated models, therefore inducing real-time constraints to make the different time scales consistent. In order to allow the real-time execution of numerical simulation models, computation times have to satisfy a well-defined model of real-time constraints.

Considering the numerical integration of Ordinary Differential Equations (ODEs), the real-time constraints can be seen as computing fast enough to be compliant with the system's dynamics to ensure that the deviations w.r.t. the ideal (continuous) plant's behavior can be counted as negligible. For example, in an engine model, the combustion in the cylinders may require tight needs in term of processor resources to cope with the fast transients of this particular phase.

In our point of view, real-time simulators for HIL validation may be considered as weakly-hard real-time systems [3] that may miss a specified number of deadlines in a specified way, in order to guarantee a level of quality of service. Men-in-the-loop simulators, on the other hand, may be considered as soft real-time systems where deadlines can be missed in a non predictable way.

This paper considers weakly hard real-time HIL systems and focuses on computation time aspects, where small enough computation times are necessary to comply with real-time constraints. First, engine models are introduced. Then, the characteristic of numerical solvers are presented through benchmark test results to show their influence on real-time simulation. Finally, model splitting and parallel computing are introduced to improve efficiency to reach real-time constraints.

2. Engine Model

The considered cyber-physical system involves an engine and its controllers. The engine represents the physical system part, it is modeled in the continuous-time domain using ODEs. It belongs to the hybrid systems category because of some discontinuous behaviors which correspond to events triggered off when a given threshold is crossed. Controllers, that supervise 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. For this case study, the real-time system is defined as if controllers interact with the engine model as though it was in a real physical system, which means that the behavior of the model is expected to be similar (or close enough) to the one of the real continuous system. In other words, some deadlines may be relaxed in a controllable way by synchronizing controllers and engine model only at some meeting points. Eventually, it must not disturb the test results.

The considered engine is a four cylinder diesel engine with fixed geometry turbocharger where the combustion process is

modeled based on Wiebe's law. The model contains 87 continuous states and 420 event indicators (of discontinuities). Each cylinder requires four strokes of its piston (two revolutions of the crankshaft) to complete the sequence of events which produces one power stroke. It comprises [5]:

- An intake stroke, which draws up fresh mixture into the cylinder from the inlet valve.
- A compression stroke, when both valves are closed and the mixture inside the cylinder is compressed to a small fraction of its initial volume. Toward the end of the compression stroke, combustion is initiated and the cylinder pressure rises very fast.
- A power or expansion stroke (combustion phase), where the high-temperature and high-pressure gases push the piston down and force the crank to rotate.
- An exhaust stroke, where the remaining burned gases exit the cylinder through the exhaust valve.

3. Numerical solvers

The need for deterministic computation times made fixed step solvers considered as the prerequisite solution for real-time simulation. This paper aims to revisit this assumption. This section will discuss solvers characteristics, more especially their step management, in order to characterize their influence on weakly-hard real-time constraints.
The main differences between numerical schemes are:

- Order, which represents the accuracy of numerical solution due to the truncation of Taylor series.
- Explicit/Implicit method: Implicit schemes are often less accurate in the beginning compared to explicit schemes. They are also more complicated to implement and require much computations at each time step because they need the resolution of a non linear system; still, they are stable mostly for stiff systems, whereas explicit schemes need to use very short time steps to ensure stability.
- Fixed/Variable (adaptive) time step: In general, the time step must be much lower or even negligible compared to system dynamic in order to have stability. Stiff problems arise when the system has several time constants that are very different (distant). For fixed time step methods, the step size must be chosen tiny enough to cope with the fastest system's transient for the whole simulation time. However the number of integration for a simulation step according to their order is bounded and so the execution time is predictable. For adaptive methods, the integration step is driven by the integration error, iterating until it reaches a predefined bound on the error, thus following the variations of the system's dynamic along the simulation.
- One-step/Multi-step method: One-step methods (e.g. Runge-Kutta) only use the current values of the solution y and its derivatives $f = dy/dt$. Multi-step methods use several past values of y (e.g. Backward Differentiation Formulas (BDFs)) and $f = dy/dt$ (e.g. Adams) to achieve a higher order of accuracy. BDF is among the most effective multi-step method for stiff systems whereas Adams method is among the best known multi-step method for solving general non-stiff systems.

The LSODAR [6] solver is based on BDFs and Adams algorithms and switch automatically between methods when the system varies from stiff to non stiff and conversely. It has also a root-finding capability to detect the events occurring during the simulation and to cleanly stop/restart the integration in case of events.

4. Benchmark tests and results

In the following benchmark tests, simulations are done with the xMOD tool. xMOD [1] is a platform that combines an integration environment for heterogeneous models together with a virtual test laboratory. The considered engine model was developed in Dymola using ModEngine library [2] and was imported in xMOD as Functional Mock-up Unit (FMU)[1]

The engine model is linked to its controller developed in Simulink thanks to the integration capabilities of xMOD. The data exchange between the model and the controller is performed periodically. This period is denoted *communication time step* and is chosen equal to 500 μs. For fixed-step solvers, there is no control on accuracy. An integration step of 50 μs is chosen in order to achieve correct simulation results. For variable-step solvers, the communication time-step is different from integration step and could even be very larger (see Figure 1). The solver automatically increase the time-step when there are no discontinuities and decreases it when a discontinuity occurs. This distinctive feature make it faster than fixed-step solvers for regular continuous system with few discontinuities. Besides, variable-step solvers are known for their accuracy and the capability to better manage numerical stability.

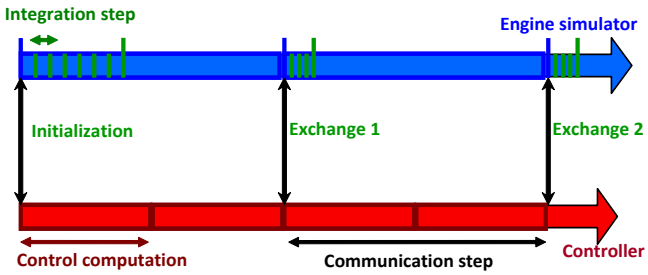

Figure 1. simulator architecture

The following tests, which constitute a work in progress, are done without controllers and don't reach currently real time ($0.5s$ simulation time correspond to $10.5s$ with Euler) due to the model complexity. Model optimization is currently being undertaken and simulation with Euler (the fast one) will be considered as reference. Tests with LSODAR show that refining the solver tolerance causes an increase in execution time. In fact, varying it from 10^{-1} to 10^{-3} increases twice the time (see Figure 2). Moreover, setting the tolerance to

[1] http://modelisar.org/

10^{-1} preserves the result correctness, so it is better to use it, especially as its execution time is close to the well known Runge-Kunta 4th order solver (RK4) (see Figure 2).

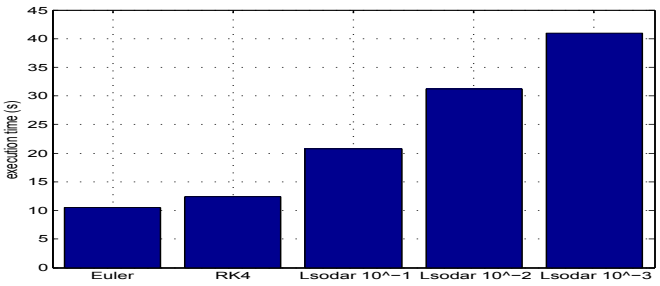

Figure 2. execution time for different solvers and tolerances

Other results using LSODAR show that the engine model presents many discontinuities. Figure 3 represents one engine cycle ($2500rpm$). This causes an interruption of the solver at every discontinuity. For this model, the interruption due to events represent $57,7\%$ of all the integration stops. So, even when the system's dynamics allows LSODAR to choose the maximum step size (500μs), the solver cannot take it and is forced to select a smaller one to fit with the discontinuity (see Figure 4). This shows that the step size is not only related to the system's dynamics but also to the occurrence of frequent events processed during the integration. We can conclude that the speed advantage of variable-step solvers decreases when the system emits a large number of events with high frequency.

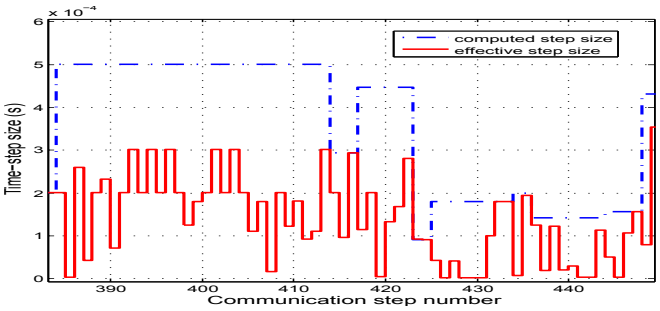

Figure 3. Impact of discontinuities on step size

Figure 5 plots LSODAR, Euler and RK4 execution times for each communication time step (500μs). LSODAR presents several peaks due to fast transients, but several times LSODAR execution times are smaller than fixed-step solvers (Euler and RK4). Besides, LSODAR execution times vary from communication step to another which enlighten the step adaptation w.r.t. the system's dynamic evolution.

Previous tests show that LSODAR is an effective variable-step solver because it can speed up when system dynamics allow it, while keeping the integration tolerance under control.

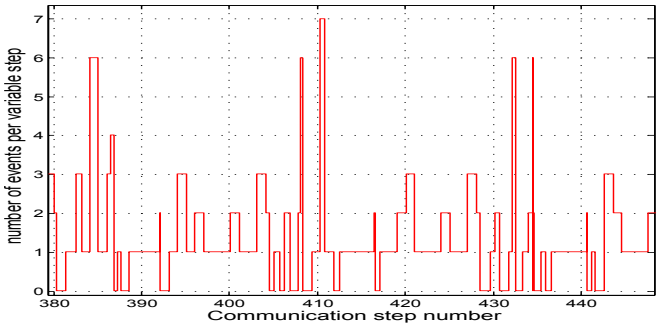

Figure 4. frequency of events on variable step

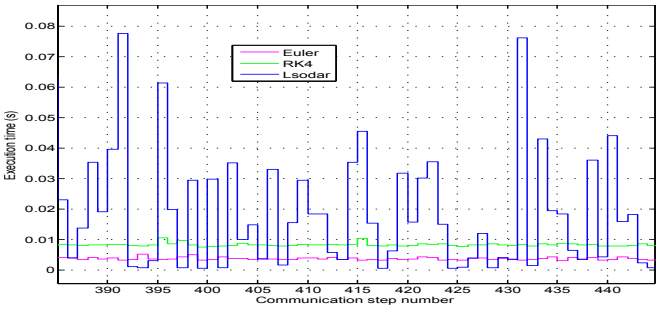

Figure 5. execution time per communication step

Although, it is not efficient when the system presents many discontinuities. On the other hand, it appears often that in a complex systems, events are only directly related to the evolution of a subset of the state vector, and that the corresponding discontinuities are independent from a physical point of view. Thus an idea is to divide the system into subsystems, to minimize useless integration interrupts due to unrelated events.

5. Model splitting to parallel computing: Waveform Relaxation

Even with sophisticated techniques, numerical solvers have reached their boundaries in speed computation terms. In order to reach real-time, the next step is to shorten even more integration time using multi-core platform by parallelizing engine model to simplify computation to the solver.

5.1. Engine model splitting

The engine model was partitioned 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 cylinder 1 to cylinder 4 in a sequential way. Then, at the end of the forth cylinder combustion, the cycle is repeated relentlessly. By splitting the models, the solver can treat locally the events during combus-

tion cycle in a single cylinder and then relax time-step until the next cycle.

With the engine model already partitioned, the aim is to apply a solver to it using the Waveform Relaxation (WR).

5.2. Waveform Relaxation

Waveform Relaxation method was introduced in [8]. It is an iterative process originating from Picard theorem, which makes possible to solve simultaneously in parallel coupled subsystems over successive time windows. Each subsystem is characterized by its waveform (i.e. its solution over a determined time interval). The purpose is to find the waveform of a subsystem, considering all the waveforms of the other subsystems constant during one iteration. For practical results ([4]), a sequential Gauss Seidel and a parallel Gauss Jacobi WR codes have been developed. The second implementation is considered in the sequel.

Given a system partitioned into two subsystems with x_1 and x_2 the two waveforms:

$\frac{dx_1}{dt} = f(t, x_1, x_2), \quad x_{10} = x_1(t = 0) \quad (1)$

$\frac{dx_2}{dt} = f(t, x_1, x_2), \quad x_{20} = x_2(t = 0) \quad (2)$

The initialization is done by freezing, during all simulation time, x_{20} for the first WR and x_{10} for the second WR. Then, the first iteration is done by integrating simultaneously (1) and (2) and saving in memory the trajectories $(x_1^1, x_1^2, .., x_1^n)$ and $(x_2^1, x_2^2, .., x_2^n)$ at the communication time-step. Each subsystem can be integrated with a variable-step or a fixed-step solver but all must use the same communication time-step.

If the tests of convergence are satisfied for iteration it: $|x^i_{1,it} - x^i_{1,it-1}| <= \epsilon$ and $|x^i_{2,it} - x^i_{2,it-1}| <= \epsilon$; for i=[1..n], then the integration is successful. Otherwise another integration is restarted, using the already computed state trajectories updated from the other subsystems, until convergence (see figure 6).

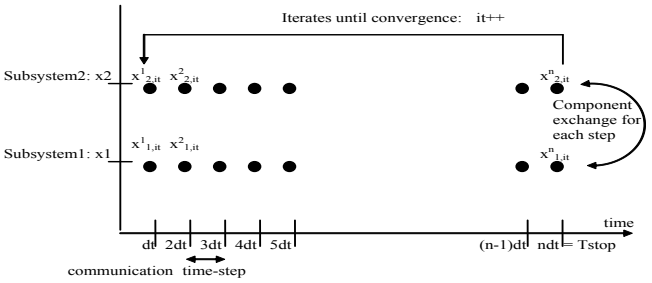

Figure 6. Waveform relaxation technique

The purpose of splitting the system and using WR method is to reduce execution time and to reach real-time constraints. With the original WR method, it is not possible because of the slowness of convergence. It is shown in [4] that longer is the simulation time until communication between the WR, slower is the convergence. In this work in progress, several parameters are studied in order to improve the number of iterations:

- Control of the solver tolerance depending on the WR iteration. In fact, for the first WR iterations, the results are not accurate because of the lack of data coming from other sub-systems. The idea is then to relax the solver tolerance just for the first iterations,then progressively tighten it.
- Initialization with infinitesimal disturbance of equilibrium position. Indeed, with initialization close to the solution, few iterations will be need to converge.
- Using *time windows* for the WR. In fact, integrating until a specified time window instead of the end of simulation eases the convergence and decreases the number of iterations.

6. Future Work: Waveform relaxation with adaptive windowing

After implementing these described techniques, WR with windowing approach could be more improved for real-time simulation by dynamically adapting the window. In fact, it is not obvious to select the optimal window size that take into account the (variable) system's dynamic behaviors. Hence, the idea, derived from variable-step solvers principle, is to compute on-line non-uniform windows whose sizes depend on the development of the waveforms. The concept is to adaptively determine the size of the next time interval based on the previously computed solution, for example the windows can be rescaled using the ratio between convergence tolerance of two successive iterations [4]. The windows size should be bounded to get a small number of WR iterations. The corresponding feedback scheduler remains to be designed.

References

[1] M. Ben Gaïd, G. Corde, A. Chasse, B. Léty, 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.

[2] 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.

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

[4] K. Burrage, C. Dyke, and B. Pohl. On the performance of parallel waveform relaxations for differential systems. *Applied Numerical Mathematics*, 20:39–55, February 1996.

[5] J. Heywood. *Internal combustion engine fundamentals*. McGraw-Hill series in mechanical engineering. McGraw-Hill, 1988.

[6] 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.

[7] 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.

[8] E. Lelarasmee, A. Ruehli, and A. 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.