2011 10th International Symposium on Distributed Computing and Applications to Business, Engineering and Science

Coupling the Parareal algorithm with the Waveform Relaxation method for the solution of Differential Algebraic Equations

Thomas Cadeau and Frédéric Magoulès
Applied Mathematics and Systems Laboratory
Ecole Centrale Paris
Châtenay-Malabry, France
Email: frederic.magoules@hotmail.com

***Abstract*—In this paper, a coupling strategy of the Parareal algorithm with the Waveform Relaxation method is presented for the parallel solution of differential algebraic equations. The classical Waveform Relaxation (in space) method and the Parareal (in time) method are first recalled, followed by the introduction of a coupled Parareal-Waveform Relaxation method recently introduced for the solution of partial differential equations. Here, this coupled method is extended to the solution of differential algebraic equations. Numerical experiments, performed on parallel multicores architectures, illustrate the impressive performances of this new method.**

***Keywords*-Differential Algebraic Equations; Domain decomposition method; Waveform Relaxation method; Parareal algorithm; Parallel and distributed computing**

I. Introduction

The solution of Differential Algebraic Equations (DAE) appears in various industrial applications such as systems or circuits stability simulation, chemical reaction, etc. For instance, in systems analysis, large interconnected networks lead to the solution of large DAE systems of hundreds of thousands of variables, while industrial softwares are currently designed for the calculation of systems of tens of thousands of variables. In circuits analysis, the same software limitations occurs, but is even more critical since DAE systems of millions of variables are needed to obtain an accurate solution. With the significant development of multiprocessors machines, the use of new algorithms designed for parallel and distributed processing appears very promising.

In this paper, after a brief presentation of the mathematical problem to solve and the linearization of the associated equations, the Waveform Relaxation in space method [1] is presented. The Parareal in time method [2] is then introduced, followed by a coupled Parareal-Waveform Relaxation method, originally introduced in [3] for Partial Differential Equations (PDE). This method is here extended to DAE in a parallel computing paradigm. This new method leads by construction to a higher speedup compared to the Waveform Relaxation in space method and the Parareal in time method. Numerical experiments, performed on parallel multicores architectures, illustrate the performance of the proposed method.

II. Differential Algebraic Equations

The mathematical system to solve is a DAE system of the form:

$$\left\{ \begin{array}{rcl} y' & = & f(y,z,t),\ t\ \in [0,T] \\ 0 & = & g(y,z,t) \end{array} \right. \tag{1}$$

where f and g are functions of the differential variables, y, the algebraic variables, z, and the time variable, t. Due to some properties of the physical problem, the functions f and g are non-linear functions, non differentiable, neither continuous. The system is considered as a rough problem, not only due to the discontinuities, but also because of its stiffness. The interesting and important point is that the functions f and g are smooth enough between two discontinuities (like in systems or circuits), which allows to consider linear approximation of f and g in such interval. Fixing the functions f_0 and g_0 at $t = 0$, the system to solve becomes the following:

$$\left\{ \begin{array}{rcl} y' & = & M\delta y + N\delta z + f_0,\ t\ \in [0,T] \\ 0 & = & P\delta y + Q\delta z + g_0, \end{array} \right. \tag{2}$$

with $\delta y = y - y_0$, $\delta z = z - z_0$, where y, z are the states at time $t \in [0,T]$, and y_0, z_0 are the states at time $t = 0$. The integration scheme used for the differential equations is a backward differentiation formula (BDF), using a trapezoidal rule. The implicit system to solve can be written as:

$$A \left(\begin{array}{c} \delta y_{n+1} \\ \delta z_{n+1} \end{array} \right) = B \left(\begin{array}{c} \delta y_n \\ \delta z_n \end{array} \right) + b \tag{3}$$

where:

$$A = \left(\begin{array}{cc} I - \frac{h}{2}M & -\frac{h}{2}N \\ P & Q \end{array} \right),$$

$$B = \left(\begin{array}{cc} I + \frac{h}{2}M & \frac{h}{2}N \\ 0 & 0 \end{array} \right), \quad b = \left(\begin{array}{c} hf_0 \\ -g_0 \end{array} \right).$$

Solving the system Eq. (3) implies a matrix-vector multiplication by the matrix B and the solution of a linear system with the matrix A. The matrices A and B, and the vector b depend only of the timestep h, chosen at the begin of the computation. Consequently, A, B and b are fixed for the simulation and do not require additional computation. Once the right-hand side (rhs) of Eq. (3) evaluated, a direct

978-0-7695-4415-1/11 $26.00 © 2011 IEEE
DOI 10.1109/DCABES.2011.34

method is used to solve the linear system with the matrix A. The factorization of the matrix A is done at the beginning of the computation and only a forward/back substitution is performed at each timestep. Finally, the solution is rebuilt knowing the initial values:

$$\begin{pmatrix} y_{n+1} \\ z_{n+1} \end{pmatrix} = \begin{pmatrix} \delta y_{n+1} \\ \delta z_{n+1} \end{pmatrix} + \begin{pmatrix} y_0 \\ z_0 \end{pmatrix}. \quad (4)$$

The following algorithm details the different steps of the sequential solver based on the lower/upper factorization and the corresponding forward/back substitution.

Lower/Upper factorization of matrix A
while $t_n < T$ **do**
  Compute the rhs, Eq. (3)
  Forward/back substitution, Eq. (3)
  Update the solution, Eq. (4)
end while

III. Waveform Relaxation Method

Domain Decomposition methods [4]–[7] are well suited for parallel computations. Indeed, the division of a problem into smaller sub-problems, through artificial subdivisions of the domain/matrix, is a means for introducing parallelism. Domain decomposition strategies include in one way or another the following ingredients: (i) a decomposer to split a domain/matrix into sub-domains/sub-matrices using different heuristics; (ii) local solvers (direct or iterative, exact or approximate) to find solutions for the sub-problems for specific boundary conditions on the interface; (iii) interface conditions [8], [9] (weak or strong) enforcing compatibility and equilibrium between overlapping or non-overlapping sub-domains/sub-matrices; (iv) a solution strategy for the interface problem [10], [11]. The differences between the methods lie in how those ingredients are actually put to work and how they are combined to form an efficient solution strategy for the problem at hand.

To ensure good performances of domain decomposition methods, one first point is to find an appropriate partitioning of the domain/matrix. The configurations to avoid are: (i) a bad distribution of the variables which leads to singular sub-matrices or which affects the convergence; (ii) a bad loadbalancing of the variables between the sub-domains/sub-matrices which causes strong unbalance computation times between the processors; (iii) large interfaces size which leads to high communications time. Given a sub-domain/sub-matrix named (s), the indexes of the variables belonging to the internal area of the sub-domain/sub-matrix or to the overlapping area of the neighboring sub-domain/sub-matrix are named (i_s). The indexes of the variables belonging to the neighboring sub-domain/sub-matrix in contact with the indexes of the variables (i_s) are named (b_s). Each sub-system s depends on the variables $(\delta_y^{(b_s)}, \delta_z^{(b_s)})^t$ which are computed by the neighboring sub-system. The global system Eq. (2) is projected on the partitioning of the domain/matrix and Dirichlet conditions are added on the interface between the sub-domains/sub-matrices. With these notations, each local sub-problem can be written as:

$$\begin{pmatrix} y^{(i_s)'} \\ 0 \end{pmatrix} = J^{(i_s)} \begin{pmatrix} \delta y^{(i_s)} \\ \delta y^{(b_s)} \\ \delta z^{(i_s)} \\ \delta z^{(b_s)} \end{pmatrix} + \begin{pmatrix} f_0^{(i_s)} \\ g_0^{(i_s)} \end{pmatrix} \quad (5)$$

where $J^{(i_s)} = \begin{pmatrix} M^{(i_s)} & M^{(b_s)} & N^{(i_s)} & N^{(b_s)} \\ P^{(i_s)} & P^{(b_s)} & Q^{(i_s)} & Q^{(b_s)} \end{pmatrix}$.

Using the same numerical scheme than for the sequential solver (algorithm described on Section II), each sub-system can be written as follows:

$$A^{(i_s)} \begin{pmatrix} \delta y_{n+1}^{(i_s)} \\ \delta z_{n+1}^{(i_s)} \end{pmatrix} = B^{(i_s)} \begin{pmatrix} \delta y_n^{(i_s)} \\ \delta z_n^{(i_s)} \end{pmatrix} + b^{(i_s)} \quad (6)$$

where $A^{(i_s)}$, $B^{(i_s)}$, $b^{(i_s)}$ are defined by:

$$A^{(i_s)} = \begin{pmatrix} I^{(i_s)} - \frac{h}{2} M^{(i_s)} & -\frac{h}{2} N^{(i_s)} \\ P^{(i_s)} & Q^{(i_s)} \end{pmatrix},$$

$$B^{(i_s)} = \begin{pmatrix} I^{(i_s)} + \frac{h}{2} M^{(i_s)} & \frac{h}{2} N^{(i_s)} \\ 0 & 0 \end{pmatrix},$$

$$b^{(i_s)} = \begin{pmatrix} h f_0^{(i_s)} \\ -g_0^{(i_s)} \end{pmatrix} + \begin{pmatrix} hM^{(b_s)} & hN^{(b_s)} \\ -P^{(b_s)} & -Q^{(b_s)} \end{pmatrix} \begin{pmatrix} \delta y_{n+1}^{(b_s)} \\ \delta z_{n+1}^{(b_s)} \end{pmatrix}.$$

Knowing $(\delta y_{n+1}^{(b_s)}, \delta z_{n+1}^{(b_s)})^t$ the sub-problem can be solved directly by the forward/back substitution. The difficulty is that $(\delta y_{n+1}^{(b_s)}, \delta z_{n+1}^{(b_s)})^t$ is not known and has to be computed by the other sub-systems. Thus, the Waveform Relaxation algorithm introduces an iterative scheme, with an iteration number k, as:

$$A^{(i_s)} \begin{pmatrix} \delta y_{n+1}^{(i_s),k+1} \\ \delta z_{n+1}^{(i_s),k+1} \end{pmatrix} = B^{(i_s)} \begin{pmatrix} \delta y_n^{(i_s),k} \\ \delta z_n^{(i_s),k} \end{pmatrix} + b^{(i_s),k} \quad (7)$$

where

$$b^{(i_s),k} = \begin{pmatrix} hM^{(b_s)} & hN^{(b_s)} \\ -P^{(b_s)} & -Q^{(b_s)} \end{pmatrix} \begin{pmatrix} \delta y_{n+1}^{(b_s),k} \\ \delta z_{n+1}^{(b_s),k} \end{pmatrix} + \begin{pmatrix} h f_0^{(i_s)} \\ -g_0^{(i_s)} \end{pmatrix}.$$

Each local sub-problem is allocated to a different processor and is solved in parallel. Communications between processes occur at each iteration since process s needs to receive $\delta y_{n+1}^{(b_s),k}$ and $\delta z_{n+1}^{(b_s),k}$. The set of these timesteps is named a window. The time interval $[0, T]$ is split into W windows $[T_w, T_{w+1}]$:

$$0 = T_0 < T_1 < ... < T_w < T_{w+1} < ... < T_W = T.$$

In this study, the window size is fixed to S timesteps. Thus $T_w = t_{S.w} = w.S.h$ and $(\delta y_{S.w}^{(i_s)}, \delta z_{S.w}^{(i_s)})^t$ is the solution at time T_w. Knowing $\delta y_n^{(b_s),k}$ and $\delta z_n^{(b_s),k}$ for $n \in \{w.S, ..., (w+1).S\}$, the process s computes on its own $\delta y_n^{(i_s),k+1}$ and $\delta z_n^{(i_s),k+1}$ for $n \in \{w.S, ..., (w+1).S\}$. The iterative scheme is now defined. To initialize the iterative

algorithm, initial boundary values need to be defined. To find coherent values without additional computational cost, the initial guess is fixed at the last known value:

$$\begin{pmatrix} \delta y_n^{(i_s),0} \\ \delta z_n^{(i_s),0} \end{pmatrix} = \begin{pmatrix} \delta y_{w.S}^{(i_s)} \\ \delta z_{w.S}^{(i_s)} \end{pmatrix},\ n \in \{w.S, ..., (w+1).S\}. \tag{8}$$

At the convergence, the solution is rebuilt, as in the sequential solver:

$$\begin{pmatrix} y_{n+1}^{(i_s)} \\ z_{n+1}^{(i_s)} \end{pmatrix} = \begin{pmatrix} \delta y_{n+1}^{(i_s)} \\ \delta z_{n+1}^{(i_s)} \end{pmatrix} + \begin{pmatrix} y_0^{(i_s)} \\ z_0^{(i_s)} \end{pmatrix} \tag{9}$$

The following algorithm details the steps of the Waveform Relaxation solver with a local solver similar to the sequential algorithm on Section II. The set of indexes (i_s, b_q) is the intersection set of the indexes i_s of the sub-domain/sub-matrix s and of the indexes b_q of the sub-domain/sub-matrix q, i.e. $i_s \cap b_q$.

Lower/Upper factorization of matrix $A^{(i_s)}$
while $t_n < T$ i.e. $w < W$ **do**
  Initialise with Eq. (8)
  while WR has not converged, iterate on k **do**
    Recv $\begin{pmatrix} \delta y_{n+1}^{(i_q,b_s),k} \\ \delta z_{n+1}^{(i_q,b_s),k} \end{pmatrix}$,
    and Send $\begin{pmatrix} \delta y_{n+1}^{(i_s,b_q),k+1} \\ \delta z_{n+1}^{(i_s,b_q),k+1} \end{pmatrix}$,
    $\forall n = \{S.w, ..., S.(w+1)\}$
    while $t_n < T_{w+1} = t_{S.(w+1)}$ **do**
      Compute the rhs, Eq. (7)
      Forward/back substitution, Eq. (7)
    end while
  end while
  Update the solution, Eq. (9)
end while

IV. Parareal in Time Method

The Parareal in time method [2] aims to split the time domain and to solve each associated sub-problem independently. The time domain $[0, T]$ is split into time windows $[T_w, T_{w+1}]$:

$$0 = T_0 < T_1 < ... < T_w < T_{w+1} < ... < T_W = T.$$

The objective is to solve the initial problem on each time window, in parallel. To ensure good load balancing, each time window contains the same number S of timestep h, i.e. $T_{w+1} - T_w = \Delta T = S.h$. Obviously, to solve the problem on $[T_w, T_{w+1}] = [t_{S.w}, t_{S.(w+1)}]$, the initial solution at time $T_w = t_{S.w}$ should be known.

Let's define $F_{\Delta T}$ a propagator which allows to compute the solution on the time window $[T_w, T_{w+1}]$, knowing the initial solution at time T_w. This propagator is based on the sequential solver, called the Fine propagator in the following.

Let's define $C_{\Delta T}$ a cheap propagator which allows to compute an approximation of the solution at time T_{w+1} knowing an approximation of the solution at time T_w. This propagator is based on the sequential solver, but with $h = \Delta T$, called the Coarse propagator in the following.

The matrix associated to $F_{\Delta T}$ (resp. $C_{\Delta T}$) is noted A_F (resp. A_C). The coarse propagator is computed sequentially, this allows to have approximation of all $(\delta y_{S.w}, \delta z_{S.w})^t$. These approximate solutions allow the fine solver to compute the solution on each time window in parallel. Then additional computation of the coarse solver, using the values computed by the fine solver, enable to introduce the iteration scheme, with an iteration number k, of the Parareal in time algorithm:

$$\begin{cases} \begin{pmatrix} \delta y_{n+1}^0 \\ \delta z_{n+1}^0 \end{pmatrix} &= C_n^0 \\ \begin{pmatrix} \delta y_{n+1}^{k+1} \\ \delta z_{n+1}^{k+1} \end{pmatrix} &= C_n^{k+1} + F_n^k - C_n^k \end{cases} \tag{10}$$

where

$$C_n^{k+1} = C_{\Delta T}(\begin{pmatrix} \delta y^{k+1} \\ \delta z_n^{k+1} \end{pmatrix}), F_n^{k+1} = F_{\Delta T}(\begin{pmatrix} \delta y_n^k \\ \delta z_n^k \end{pmatrix}).$$

The following algorithm details a parallel implementation of the Parareal in time method. Note that for the first process (called 0), *Recv* is not a reception from another process but just the initialization from the state at t_0. Note also that *Send* of the last process is not used. Each computation of $F_{\Delta T}()$ and $C_{\Delta T}()$ uses forward/back substitution, to solve Eq. (3).

Lower/Upper factorization of A_C
Lower/Upper factorization of A_F
Recv $\begin{pmatrix} \delta y_{S.w}^0 \\ \delta z_{S.w}^0 \end{pmatrix}$ form previous process
Compute $\begin{pmatrix} \delta y_{S.(w+1)}^0 \\ \delta z_{S.(w+1)}^0 \end{pmatrix} = C_{S.w}^0$
Send $\begin{pmatrix} \delta y_{S.(w+1)}^0 \\ \delta z_{S.(w+1)}^0 \end{pmatrix}$ to following process
while Parareal has not converged, iterate on k **do**
  Compute $\begin{pmatrix} \delta \hat{y}_{S.(w+1)}^k \\ \delta \hat{z}_{S.(w+1)}^k \end{pmatrix} = F_{S.w}^k$
  Recv $\begin{pmatrix} \delta y_{S.w}^{k+1} \\ \delta z_{S.w}^{k+1} \end{pmatrix}$ form previous process
  Compute $\begin{pmatrix} \delta \tilde{y}_{S.(w+1)}^{k+1} \\ \delta \tilde{z}_{S.(w+1)}^{k+1} \end{pmatrix} = C_{S.w}^{k+1}$
  Update $\begin{pmatrix} \delta y_{S.(w+1)}^{k+1} \\ \delta z_{S.(w+1)}^{k+1} \end{pmatrix}$ with Eq. (10)
  Send $\begin{pmatrix} \delta y_{S.(w+1)}^k \\ \delta z_{S.(w+1)}^k \end{pmatrix}$ to following process
end while
Update the solution, Eq. (4)

V. Coupling Parareal and Waveform Relaxation Methods

The Waveform Relaxation method and the Parareal in time method have a limited speedup. Indeed, on one hand, for the Waveform Relaxation method, the solution cost of the local sub-domains should be high and the communications between the sub-domains cost should be small. It is thus important to have a large number of sub-domains, in such a way that the number of variables within each sub-domain is large enough. In other words, this means that to split the initial domain into sub-domains with few variables within each sub-domain leads to a lower efficiency. On the other hand, the Parareal in time method has a speedup bounded by the number of processes divided by two. Coupling these two methods [3] allows to combine the good speedup of the Waveform Relaxation, with the maximum possible number of sub-domains i.e. before its speedup decreases, and to continue to increase its speedup by using the Parareal in time method.

The coupled Parareal-Waveform Relaxation method is based on the Parareal in time method. The Fine propagator $F_{\Delta T}$ is based on the Waveform Relaxation method. With the same time windows than the Parareal in time algorithm $[T_w, T_{w+1}]$, and with the same timestep h than the sequential method. The Coarse propagator $C_{\Delta T}$ is based on the same method, and with both the timestep and the window size equal to the Parareal time windows: $h = T_{w+1} - T_w$. There is no reason for the Fine and the Coarse propagators to converge at each iteration of the Parareal method. They only need to converge at the last iteration of the Parareal method. This is the reason why the number of iterations of the propagators is bounded to a small number, 2 or 3. These new propagators with limited number of iterations are denoted $\tilde{F}_{\Delta T}$ and $\tilde{C}_{\Delta T}$.

The working variables in each Waveform Relaxation are written as follows: $u_{n,k}^{(i_s),l}$ with s the sub-domain number, n the index of the timestep, k and l the iteration numbers of the Parareal and Waveform Relaxation propagator respectively. The Parareal solutions are written as follows: $u_{n,k}^{(i_s)}$. The difference with the Parareal Algorithm (Section IV) is that the communication *Send/Recv* are done from the previous and to the following processes group or processes communicator. Each communicator computes the Fine and Coarse propagator in parallel following the algorithm described on Section III. The Fine Waveform Relaxation initial guess of k^{th} coupled Parareal-Waveform Relaxation is not set to a constant initialization but to an improved approximation which does not require additional computational cost, $n \in \{w.S, ..., (w+1).S\}$:

$$\begin{pmatrix} \delta y_{n,k+1}^{(i_s),0} \\ \delta z_{n,k+1}^{(i_s),0} \end{pmatrix} = \begin{pmatrix} \delta y_{w.S,k+1}^{(i_s)} \\ \delta z_{w.S,k+1}^{(i_s)} \end{pmatrix} + \tilde{F}_{n,k}^{(i_s)} - \begin{pmatrix} \delta y_{w.S,k}^{(i_s)} \\ \delta z_{w.S,k}^{(i_s)} \end{pmatrix}. \quad (11)$$

The following algorithm shows a parallel implementation of the coupled Parareal-Waveform Relaxation method using the Waveform Relaxation algorithm introduced in Section III and initialized by Eq.(11).

Lower/Upper factorization of A_C
Lower/Upper factorization of A_F
Recv $\begin{pmatrix} \delta y_{S.w}^{0} \\ \delta z_{S.w}^{0} \end{pmatrix}$ form previous process
Compute $\begin{pmatrix} \delta y_{S.(w+1)}^{0} \\ \delta z_{S.(w+1)}^{0} \end{pmatrix} = \tilde{C}_{S.w}^{0}$ by WR
Send $\begin{pmatrix} \delta y_{S.(w+1)}^{0} \\ \delta z_{S.(w+1)}^{0} \end{pmatrix}$ to following process
while Parareal has not converged, iterate on k **do**
  Compute $\begin{pmatrix} \delta \hat{y}_{S.(w+1)}^{k} \\ \delta \hat{z}_{S.(w+1)}^{k} \end{pmatrix} = \tilde{F}_{S.w}^{k}$ by WR
  Recv $\begin{pmatrix} \delta y_{S.w}^{k+1} \\ \delta z_{S.w}^{k+1} \end{pmatrix}$ form previous process
  Compute $\begin{pmatrix} \delta \tilde{y}_{S.(w+1)}^{k+1} \\ \delta \tilde{z}_{S.(w+1)}^{k+1} \end{pmatrix} = \tilde{C}_{S.w}^{k+1}$ by WR
  Update $\begin{pmatrix} \delta y_{S.(w+1)}^{k+1} \\ \delta z_{S.(w+1)}^{k+1} \end{pmatrix}$ with Eq. (10)
  Send $\begin{pmatrix} \delta y_{S.(w+1)}^{k} \\ \delta z_{S.(w+1)}^{k} \end{pmatrix}$ to following process
end while
Update the solution, Eq. (4)

VI. Numerical Results

The problem considered in this section consists to solve a DAE system composed of 100.000 differential and algebraic variables, during a physical simulation time equal to 6.4 seconds. This time corresponds to the time between two events, i.e. where the linearization of the functions f and g is valid. Despite the space dimension of the DAE system might looks small, the reader should keep in mind that after the time discretization, this problem is impossible to solve in a near real time simulation without parallel computing.

To solve this problem, the system (in space) is split into 2, 4, 8, 16, 32, and 64 sub-systems with a mesh partitioning software [12]–[14]. In this paper, the METIS software has been used. The Waveform Relaxation method is applied on these sub-systems. Each sub-system is allocated to one different processor. As illustrated in Figure 1, the speedup of the Waveform Relaxation method increases till 16 sub-systems. After reaching a peak, considering more sub-systems does not increase the speedup, but decrease it. This is coherent with the analysis performed in the previous sections, since when more than 16 sub-systems are considered on this problem (100.000 variables), each sub-system contains too few variables to ensure a good efficiency of the method. In order to improve the speed-up, the coupled Parareal-Waveform Relaxation method is

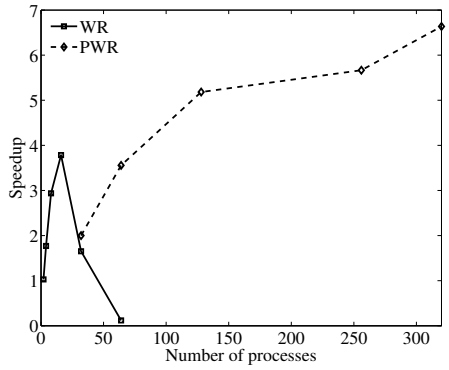

Figure 1. Comparison of the speedup of the Waveform Relaxation method and the coupled Parareal-Waveform Relaxation method.

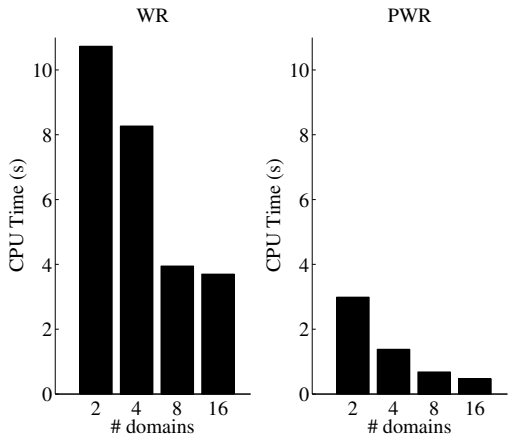

Figure 2. Comparison of the computational time of the Waveform Relaxation method and the coupled Parareal-Waveform Relaxation method.

now applied. For this purpose, the system (in space) is split into 16 sub-systems (in space) and into several windows (in time), forming a space-time grid. Each windows always contains 10 timesteps. Each sub-system, i.e. a cell of this space-time grid, is allocated to one different processor. The coupled Parareal-Waveform Relaxation method illustrated in Figure 1 shows that the speedup continues to increase when considering more processors.

The results reported Figure 2 illustrate the respective computational time of the Waveform Relaxation method and the coupled Parareal-Waveform Relaxation method upon the number of sub-systems (in space) when the system is split into 32 windows (in time). This figure clearly shows the extraordinary reduction of the computational time when using the coupled Parareal-Waveform Relaxation method.

VII. Conclusion

In this paper a new coupled Parareal-Waveform Relaxation method has been presented for solving Differential Algebraic Equations. This method outperforms the existing Waveform Relaxation method and the Parareal in time methods, by taking benefits of these two methods. The numerical results obtained illustrate the impressive efficiency of this coupled Parareal-Waveform Relaxation method, which allows to take full advantage of parallel multicores architectures.

References

[1] J. White and A. Sangiovanni-Vincentelli, *Relaxation Techniques for the Simulation of VLSI Circuits*. Kluwer Academic Publishers, 1987.

[2] J.-L. Lions, Y. Maday, and G. Turinici, "Résolution d'EDP par un schéma en temps pararéel," *Comptes Rendus de l'Académie des Sciences - Série I - Mathematics*, vol. 332, no. 7, pp. 661–668, 2001.

[3] R. Gueta and Y. Maday, "Coupling the Parareal algorithm with overlapping and nonoverlapping domain decomposition methods," ser. Lecture Notes in Computational Science and Engineering. Springer, (in press).

[4] B. Smith, P. Bjorstad, and W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, UK, 1996.

[5] A. Quarteroni and A. Valli, *Domain Decomposition Methods for Partial Differential Equations*. Oxford University Press, Oxford, UK, 1999.

[6] F. Magoulès and F.-X. Roux, "Lagrangian formulation of domain decomposition methods: a unified theory," *Applied Mathematical Modelling*, vol. 30, no. 7, pp. 593–615, 2006.

[7] A. Toselli and O. Widlund, *Domain Decomposition methods: Algorithms and Theory*. Springer, 2005.

[8] Y. Maday and F. Magoulès, "Absorbing interface conditions for domain decomposition methods: a general presentation," *Computer Methods in Applied Mechanics and Engineering*, vol. 195, no. 29–32, pp. 3880–3900, 2006.

[9] P. Collino, G. Delbue, P. Joly, and A. Piacentini, "A new interface condition in the non-overlapping domain decomposition," *Computer Methods in Applied Mechanics and Engineering*, vol. 148, pp. 195–207, 1997.

[10] Y. Maday and F. Magoulès, "Optimal convergence properties of the FETI domain decomposition method," *International Journal for Numerical Methods in Fluids*, vol. 55, no. 1, pp. 1–14, 2007.

[11] M. Gander, L. Halpern, F. Magoulès, and F.-X. Roux, "Analysis of patch substructuring methods," *International Journal of Applied Mathematics and Computer Science*, vol. 17, no. 3, pp. 395–402, 2007.

[12] G. Karypis and V. Kumar, *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0*, 1998.

[13] C. Walshaw, *The serial JOSTLE librairy user guide : Version 3.0*, 2002.

[14] F. Pellegrini, *Scotch and LibScotch 5.1 User's Guide*, 2008.