A Schur Complement Method for DAE/ODE Systems in Multi-Domain Mechanical Design

David Guibert[1] and Damien Tromeur-Dervout[1,2*]

[1] CDCSP - ICJ UMR5208/CNRS Université Lyon 1 F-69622 Villeurbanne Cédex
[2] INRIA/IRISA/Sage Campus de Beaulieu F-35042 Rennes Cédex
{dguibert,dtromeur}@cdcsp.univ-lyon1.fr

Summary. The large increase of unknowns in multi-domain mechanical modeling leads to investigate new parallel implementation of ODE and DAE systems. Unlike space domain decomposition, no geometrical information is given to decompose the system. The connection between unknowns have to be built to decompose the system in subsystems. A Schur Complement DDM can then be applied. During some time steps, the Jacobian matrix can be frozen allowing to speed-up the Krylov solvers convergence by projecting onto the Krylov subspace. This kind of DAE are stiff and the numerical procedure needs special care.

1 Introduction

Problems coming from the multi-domain mechanical design lead to solve systems of Ordinary Differential Equations (ODE) or Differential Algebraic Equations (DAE). Designers of mechanical systems want to achieve realistic modeling, taking into account more and more physics. Mathematically speaking, these features lead to have DAE/ODE systems with a large number of unknowns. Moreover these systems are usually stiff and eventually exhibits some discontinuities. So robust solvers with adaptive time stepping strategy must be designed.

Up to now, the main approach to obtain an ODE system parallel solver uses the parallelizing "across the method" of the time integrator. Runge-Kutta methods involve several stages. The aim of this kind of parallelization is to compute each stage of the method on a dedicated processor ([2, 1, 9, 3]).

This kind of parallelization is very limited. The number of processors involved can only be equal to the number of stages of the method.

We have shown in ([3]) that a speed-up nearby 2.5 can be obtained on 3 processors using Radau IIa method (a 3-stage RK method, see [4, 5] for more details).

In this paper, we propose a new approach based on the Schur Complement Method in order to decompose the system into smaller systems. In the Partial Differential Equation framework, the decomposition is given by the geometrical data

* Funded: National Research Agency: technologie logicielle 2006-2009 PARADE Project Mathgrid and cluster ISLES project HPC Région Rhône-Alpes

and the order of discretization scheme. Conversely in the DAE/ODE framework, no a priori knowledge of the coupled variables is available. This is the main issue to be solved.

We will show in section 2 how a Schur complement method can be implemented in the resolution of an ODE system. A brief description of the LSODA integrator will be given. Then in section 3, a strategy is applied to extract automatically the dependencies between the variables. These dependencies are viewed as an adjacency matrix and then, as in spatial domain decomposition, classical partitioning tools can be used. The algorithm is explained in section 4 and some numerical results are shown in section 5.

2 Differential Integrators

An initial value problem is considered,

for an ODE

$$(P_{ODE}) \begin{cases} \dfrac{dy}{dt} = f(t, y(t)), \\ y(t_0) = y_0, \end{cases} \tag{1}$$

or for a DAE

$$(P_{DAE}) \begin{cases} F(t, y(t), y'(t)) = 0, \\ y(t_0) = y_0, \\ y'(t_0) = y'_0. \end{cases} \tag{2}$$

The problem is assumed to be stiff. To solve the problem (P), a "predictor-corrector" scheme may be used. The main idea of such solver is to build a prediction $y_{n(0)}$ of the solution from a polynomial fit. Then the prediction is corrected by solving a nonlinear system

$$G_{ODE}(y_n) = y_n - \beta_0 h_n f(t_n, y_n) - \sum_{i>0}^{k} \alpha_{n,i} y_{n-i} = 0, \tag{3}$$

$$G_{DAE}(y_n) = F\left(t_n, y_n, h_n^{-1} \sum_{i=0}^{q} \alpha_{n,i} y_{n-i}\right) = 0, \tag{4}$$

where β_0 is a constant given by the integration scheme and h_n the current time step and $\alpha_{n,i}$ are the parameters of the method in use.

This means that the solution y_n at the time t_n is computed as follows.

- A predicted value $y_{n(0)}$ is computed to be used as initial guess for the nonlinear iteration (where $\alpha^p_{n,i}$ and β^p_0 are parameters of the prediction formula):

$$y_{n(0)} = \sum_{i=1}^{k} \alpha_i^p y_{n-i} + \beta_0^p h_n \frac{dy_{n-1}}{dt}. \tag{5}$$

- Correction step (by Newton iterations)

$$\begin{cases} \left(\frac{\partial G}{\partial y}(y_{n(m)})\right) \delta y_n = -G(y_{n(m)}), \\ y_{n(m+1)} = y_{n(m)} + \delta y_n \end{cases} \tag{6}$$

with

$$\frac{\partial G_{ODE}}{\partial y} = I - \gamma \frac{\partial f}{\partial y} = I - \gamma J, \tag{7}$$

$$\frac{\partial G_{DAE}}{\partial y} = \frac{\partial F}{\partial y} + \alpha \frac{\partial F}{\partial y'} = J, \tag{8}$$

where J is the Jacobian matrix, $\gamma = \beta_0 h_n$ and $\alpha = \alpha_{n,0} h_n^{-1}$. At the end of the iteration process $y_n = y_{n(m+1)}$.

We want to apply the Schur complement method to this linear system. In domain decomposition in space, regular data dependencies are inherent to the spatial discretization scheme, which enables a relatively easy introduction of the Schur complement. The interface nodes are on the physical junction between the subdomains. In DAE/ODE, there is no regular data dependencies (even by renumbering locally the unknowns). In the considered problems, the coupling are embedded in the whole function f which is not —necessarily— explicitly known. Indeed, the function f is composed by several sub-models which are sometimes hidden (too complex, or used as black boxes). Hence a decomposition of the matrix is far from trivial to implement.

3 Automatic Partitioning of DAE/ODE Systems

In this section, we propose a method to partition *automatically* the unknowns of the dynamic system. We assume that the function f is composed by some subfunctions f_i seen as black boxes. This means that for a function f_i only its outputs and inputs are known. Let us illustrate this on the following example

$$\begin{cases} \dfrac{dy_1}{dt} = f_1(y_1, y_2, y_4), \\ \dfrac{dy_2}{dt} = f_2(y_1, y_3), \\ \dfrac{dy_3}{dt} = f_3(y_3, y_4), \\ \dfrac{dy_4}{dt} = f_4(y_3, y_4). \end{cases} \tag{9}$$

A modification of the variables y_1 and/or y_3 may change the value of the output of f_2, the time derivative of y_2.

The coupling between the variables and their derivatives can be summarized into an incidence matrix (see the graph theory for example in [6])

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}. \tag{10}$$

The value 1 can be viewed as a dependence between two nodes in the computation of one column of the Jacobian matrix.

Having this pattern, we know that in graph theory formulation, the reduction of the coupling between the nodes of a graph is done by minimizing of the number of edges cut in the graph. A graph partitioning tool such as [7] is used.

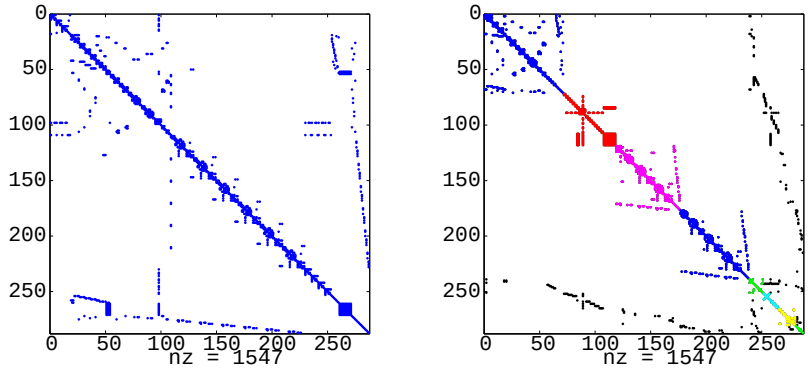

Fig. 1. Example of the Jacobian matrix of a V10 engine pump problem. Initial pattern on top and the pattern using 4 partitions

4 Algorithm

Now we concentrate on the algorithm to solve the linear system. The first step was the construction of the pattern of the Jacobian matrix (i.e. the incidence matrix corresponding to the interaction between the variables). The use of a graph partitioning tool decouples the system in sub-systems, separating the variables in internal variables and interface variables (those that need the values of variables belonging to another subdomain).

Given a partition, consider a doubly bordered block diagonal form of a matrix $A = I - \gamma J$ or $A = J$ (provided by the integrator)

$$A = \begin{pmatrix} B_1 & & & F_1 & \cdots & 0 \\ & \ddots & & \vdots & \ddots & \vdots \\ & & B_N & 0 & \cdots & F_N \\ E_1 & & & C_{11} & \cdots & C_{1N} \\ & \ddots & & \vdots & \ddots & \vdots \\ & & E_N & C_{N1} & \cdots & C_{NN} \end{pmatrix} = \begin{pmatrix} B & F \\ E & C \end{pmatrix}. \tag{11}$$

Locally on each subdomain one has to solve:

$$\begin{pmatrix} B_i & F_i \\ E_i & C_{ii} \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \neq i} C_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \tag{12}$$

We assume that B_i is not singular. Then

$$x_i = B_i^{-1}(f_i - F_i y_i). \tag{13}$$

Upon substituting a reduced system is obtained:

$$S_i y_i + \sum_{j \neq i} C_{ij} y_j = g_i - E_i B_i^{-1} f_i \text{ with } S_i = C_{ii} - E_i B_i^{-1} F_i. \tag{14}$$

Multiplying by S_i^{-1}, one obtain the following preconditioned reduced system for the interface

$$\begin{pmatrix} I & S_1^{-1}C_{12} & \cdots & S_1^{-1}C_{1N} \\ S_2^{-1}C_{21} & I & \cdots & S_2^{-1}C_{2N} \\ \vdots & & \ddots & \vdots \\ S_N^{-1}C_{N1} & \cdots & S_N^{-1}C_{NN-1} & I \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} \hat{g_1} \\ \vdots \\ \hat{g_N} \end{pmatrix}. \tag{15}$$

A solution method involves four steps:

- Obtain the right hand side of the preconditioned reduced system

$$\hat{g_i} = S_i^{-1}\left(g_i - E_i B_i^{-1} f_i\right). \tag{16}$$

- "Form" the Schur complement matrix.
  - A LU decomposition of the matrix without pivoting gives the LU decomposition of the matrix S_i

$$\begin{pmatrix} B_i & F_i \\ E_i & C_{ii} \end{pmatrix} = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix}. \tag{17}$$

- Solve the preconditioned reduced system.
- Back-substitute to obtain the other unknowns (fully parallel step).

4.1 Resolution of the Reduced System

The reduced system is solved by an iterative solver. The iterative solver only needs to define the matrix-vector action.

For the iterative scheme we use the generalized conjugate residual (GCR) method (as in [8]). The GCR method is described for a system of the form $Ax = b$.

1. Compute $r_0 = b - Ax_0$. set $p_0 = r_0$.
2. For $j = 0, 1, 2, ...,$ until convergence do:
   a) $\alpha_j = \frac{(r_j, Ap_j)}{(Ap_i, Ap_j)}$
   b) $x_{j+1} = x_j + \alpha_j p_j$
   c) $r_{j+1} = r_j - \alpha_j A p_j$
   d) compute $\beta_{ij} = -\frac{(Ar_{j+1}, Ap_j)}{(Ap_i, Ap_i)}$, for $i = 0, 1, ..., j$
   e) $p_{j+1} = r_{j+1} + \sum_{i=0}^{j} \beta_{ij} p_i$
   end do

Additionally to the vectors $\{p_j\}_{j=1}^k$, which are $A^T A$-orthogonal, extra vectors $\{Ap_j\}_{j=1}^k$ have to be stored. Since the projection of b onto the space $A\mathcal{K}$ with $\mathcal{K} = \text{span}\{p_j\}_{j=1}^k$ is equal to

$$\sum_{j=1}^{k} \frac{(b, Ap_j)}{(Ap_j, ap_j)} Ap_j, \tag{18}$$

the projection of the solution $x = A^{-1}b$ onto $\mathcal{K}$ is

$$\hat{x} = \sum_{j=1}^{k} \frac{(b, Ap_j)}{(Ap_j, ap_j)} Ap_j. \tag{19}$$

This observation implies that the projection onto the accumulated Krylov subspace may compute the unknown solution quite easily (involving k scalar products). Table 1 exhibits that the numerical speed-up is nearby of 15%.

Table 1. Numerical speed-up of the GCR method using the projection onto the accumulated Krylov subspace on the V10 engine pump problem

Krylov projection	#proc	CPU time	numerical speed-up
no	4	1750	1
yes	4	1515	1.15

5 Some Numerical Results

The speed-up obtained is quite good as shown in Table 2 by the elapsed times for solving the previous V10 engine pump problem. The partition number is increased from 1 to 4. One processor is used to solve one subdomain problem.

Table 2 exhibits a speed-up higher on 3 processors using this Schur DDM approach than using the parallelizing across the method. But with the Schur DDM that has been proposed here, the number of processors (which is equal to the partition number) is only limited by parallel performance considerations. For the small case considered here with 387 unknowns, the optimum partition number is 4 (see 3).

Table 2. Speed-up obtained on the V10 engine pump problem

#proc	CPU time	speed-up	#Jac	#discont	#steps
1	6845	1	65355	1089	311115
2	4369	1.56	66131	1061	315357
3	1820	3.76	65787	1059	313064
4	1513	4.52	65662	1043	313158

Table 3. Percentage of interface unknowns with respect to then number of processors $\frac{ne}{ne+\frac{n-ne}{np}}$ on the V10 engine pump problem ($n = 287$ unknowns).

#proc (np)	1	2	3	4	8	16
#interface unknowns (ne)	0	21	31	47	80	126
ratio of interface (%)	0	13	26	43	75	92

This limitation comes from the ratio between the number of interface unknowns and the computing complexity to solve subdomain problems. For the test case under consideration, the speed-up is supra-linear, because of two effects. The first one is a smaller full LU decomposition locally (i.e. on each processor). The second one is the parallelizing of the resolution on each subdomain that fits in the cache memory. Nevertheless, we expect only a linear speed-up when a sparse LU decomposition will be applied.

6 Conclusion

A Schur domain decomposition method has been investigated to solve systems of ordinary/algebraic differential equations. Because the data dependencies are not regular, an automatic process has been developed to separate the unknown variables in interface unknown variables and subdomain internal unknown variables. This approach was absolutely needed because the function $f(t, y)$ is given as a black box with only knowledge on the input variables and on the components of f to be affected. The condition number of the linear systems involved in the time integrator required a preconditioned linear Krylov solver. Some techniques to reuse computed information to speed-up the convergence have been investigated and save some elapsed-time. Next works will investigate some numerical tools to reuse computed information when some parts of the system become non-active or active during the cycle of simulation. Some questions are still open: what can be the numerical criterion to reuse the Krylov subspace when some dynamical systems situation reappears? May it be possible to use reduced systems obtained by proper orthogonal decomposition to model the interactions of other sub-systems to one given sub-system in the Schur DDM. This is the kind of question that will be addressed in the framework of the "PARallel Algebraic Differential Equations" ANR project.

References

[1] K. Burrage and H. Suhartanto. Parallel iterated method based on multistep Runge-Kutta of Radau type for stiff problems. *Adv. Comput. Math.*, 7(1-2):59–77, 1997. Parallel methods for ODEs.

[2] K. Burrage and H. Suhartanto. Parallel iterated methods based on multistep Runge-Kutta methods of Radau type. *Adv. Comput. Math.*, 7(1-2):37–57, 1997. Parallel methods for ODEs.

[3] D. Guibert and D. Tromeur-Dervout. Parallel adaptive time domain decomposition for stiff systems of ODE/DAE. *Computers & Structures*, 85(9):553–562, 2007.

[4] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations. I—Nonstiff Problems*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1993.

[5] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations. II—Stiff and Differential-Algebraic Problems*, volume 14 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 1996.

[6] P. Hansen and D. de Werra, editors. *Regards sur la Théorie des Graphes*, Lausanne, 1980. Presses Polytechniques Romandes.

[7] Metis. http://glaros.dtc.umn.edu/gkhome/views/metis. Karypis Lab.

[8] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2003.

[9] 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(1):111–127, 1990.