Domain Decomposition Methods for Stiff ODEs/DAEs.

D. Guibert D. Tromeur-Dervout

CDCSP/ICJ UMR 5208 U.Lyon1-CNRS

September, 7-9th 2006

PMAA 06 - Rennes

Outline

Outline

Motivation of DE solver parallelisation

As 0D-modelling of Complex mechanical systems leads to solve ODE or/and DAE systems with

- a large number of unknowns
  (up to 10,000 state variables + algebraic relations).
- large stiffness.
- eventually discontinuities.

These features need to have :

- a robust solver.
- an adaptive time step solver
  (to circumvent the stiffness).
- a fast solver to deliver the solution
  (real speed up compared to the best sequential solver).

But No distance limited coupling between the unknowns as in FE or FD methods used for PDE $\Rightarrow$ less easy parallelisation.

Three approaches to parallelise ODE solvers

- Parallelising "*across the method*" (K. Burrage) which distributes to the processors the computation of steps of multi-step methods as Runge-Kutta RK(4) method
- *Schur Decomposition* which automatically distributes the unknowns of the differential system to the processors.
- *Time decomposition method*
  - Multiple Shooting Methods
    - "Parareal" scheme (J.L. Lions, Y. Maday, G. Turinici,00),
    - "Pita" scheme (Ch. Farhat and M. Chandesris, 03).
    - "Multiple shooting method" (J. Stoer & R. Burlish 80, Deuflhard 74).
  - Pipelined Deferred Correction.

Outline

Parallelising across the method

Description

Assume to be solved the following Cauchy problem

$$\begin{cases} y' = f(t, y) \\ y(t_0) = y_0 \end{cases}$$

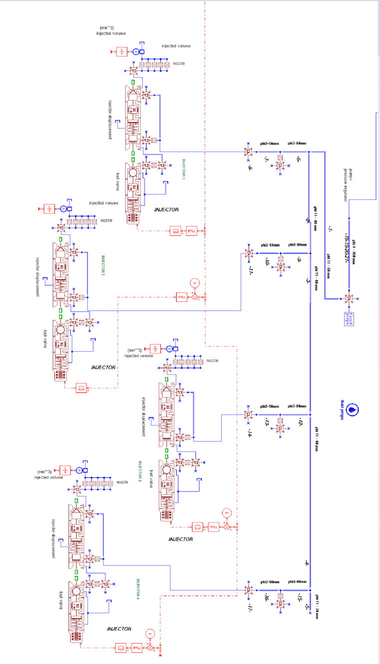
A s-stage Runge-Kutta method can be written as :

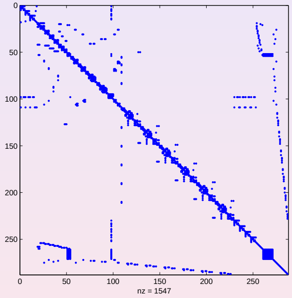
$$\begin{cases} Z = (A \otimes I_s) hF(Z) \\ y_1 = y_0 + \sum d_i z_i \end{cases} \tag{1}$$

- Parallelisation according to the structure of the tensor product.
  - $A \otimes I_s$ implemented as $I_s \otimes A$
  - $\longrightarrow$ hence we have to compute $I_s \otimes (AhF(Z))$
  - the s computations of $AhF(Z)$ are done in parallel.

Parallelising across the method

Example : a V10Injection problem





Parallelising across the method

Results

time (s)	séquential	parallel	speed up
comp. jacobian		260	
comm. jacobian		28.6	
total jacobian	847	288.6	2.93
comp. stages		43.5	
comm. stages		44.7	
total stages	143	88.2	1.62
total execution	1082	436	2.48

TAB.: Comparison between the sequential version and the parallel version (on 3 processors) of parallelised radau5 solver

⇒ But with this kind of parallelisation, the number of processors is limited by the stage number of the method.

Parallelising across the method

Results

time (s)	séquential	parallel	speed up
comp. jacobian		260	
comm. jacobian		28.6	
total jacobian	847	288.6	2.93
comp. stages		43.5	
comm. stages		44.7	
total stages	143	88.2	1.62
total execution	1082	436	2.48

TAB.: Comparison between the sequential version and the parallel version (on 3 processors) of parallelised radau5 solver

$\Rightarrow$ But with this kind of parallelisation, the number of processors is limited by the stage number of the method.

Outline

LSODA Integrator
Definition

- LSODA is notable for its ability to automatically switch between stiff and non-stiff integration.
  - $\Rightarrow$ The Suite of Nonlinear and Differential/Algebraic Equation Solvers (SUNDIALS) (CVODE).
- It solves any ODE written in its Cauchy form :

$$\begin{cases} \dfrac{\partial y}{\partial t} = f(t, y) \\ y(t_0) = y_0 \end{cases}$$

- The stiff integrator uses a ”predictor-corrector” scheme.
  - A prediction $\tilde{y}$ can be computed from a polynomial fit of the previous values $y_{n-1}, \ldots, y_{n-k}$.
  - This prediction is corrected by solving for u a nonlinear system :

$$G(u) = u - \tilde{y} - \gamma \left[f(t_m, u) - \frac{\partial \tilde{y}}{\partial t} \right] = 0$$

where γ is a constant calculated by the integrator.

LSODA Integrator
Newton

- A Newton's method is used to solve $G(u) = 0$.
- The application of Newton's method requires

$$\left(\frac{\partial G}{\partial t}\right)(u)^{-1}\delta u = (I - \gamma J)\delta u = b$$

to be solved, where J is the Jacobian matrix.

$\Rightarrow$ J can have *structural changes* due to idle subsytems $\Rightarrow$ No constant pattern during the simulation avoids symbolic factorisation.

$\Rightarrow$ The idea is to use the Schur Complement to solve this linear system.

$\Rightarrow$ But how can we decompose the unknowns of an ODE systems ?

$\Rightarrow$ But there is no (trivial) space decomposition as in PDE problems.

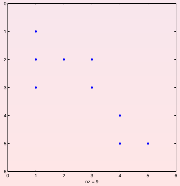
Strategy

Consider the system

$$\begin{cases} \dot{y}_1 = f_1(y_1) \\ \dot{y}_2 = f_2(y_1, y_2, y_3) \\ \dot{y}_3 = f_3(y_1, y_3) \\ \dot{y}_4 = f_4(y_4) \\ \dot{y}_5 = f_5(y_4, y_5) \end{cases} \tag{2}$$

- To study the coupling of the variables, the functions f_i are viewed as *black-boxes*.
- The input values may influence the derivatives
- According to the graph theory, the *Jacobian matrix* is viewed as an adjacency matrix.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

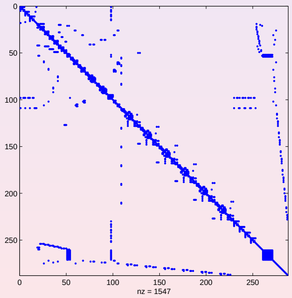


Graph partitioning tools

metis

- We want to minimize the *number of couplings* between the unknowns of different subdomains.
- In graph theory formulation, the reduction is done by a *minimisation of the number of edge cuts* in the graph.

⇒ *Metis* has been used to do this task.



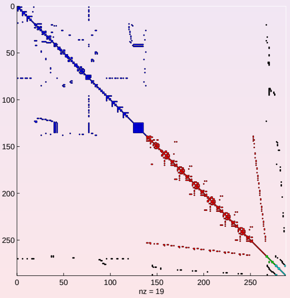


FIG.: Example : Jacobian matrices to the *V10Injection problem* with 287 unknowns (left : original pattern, right : after partitioning

Graph partitioning tools
example 3/3

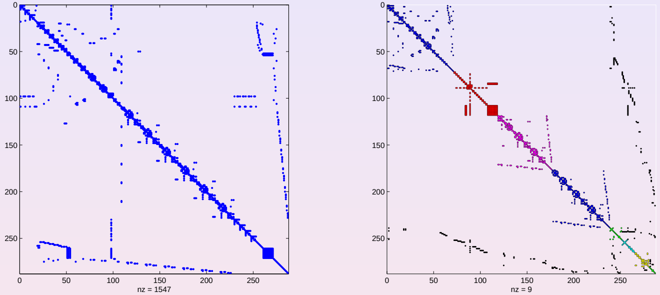


FIG.: Example : Jacobian matrices to a problem with 287 unknowns on 4 processors (left : original pattern, right : after partitioning into 4 partitions)

Schur Decomposition
Algebraic Point Of View

We consider a Doubly Bordered Block Diagonal (DBBD) form of a matrix A.

$$A = \begin{pmatrix} B_1 & & & F_1 & \cdots & 0 \\ & \ddots & & \vdots & \ddots & \vdots \\ & & B_N & 0 & \cdots & F_N \\ E_1 & & & C_{11} & \cdots & C_{1N} \\ & \ddots & & \vdots & \ddots & \vdots \\ & & E_N & C_{N1} & \cdots & C_{NN} \end{pmatrix} = \begin{pmatrix} B & F \\ E & C \end{pmatrix} \quad (3)$$

Locally have to be solved :

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

Interpretation

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

- x_i is the subvector of unknowns that are interior to the subdomain i.
- y_i is the subvector of interface unknowns of subdomain i.
- F_i is the subdomain to interface coupling seen from the subdomains.
- E_i is the interface to subdomain coupling seen from the interface.
- C_{ij} is the interface i to interface j coupling seen from the interface i.

Schur Decomposition
Resolution

We assume that B_i is not singular.

$$x_i = B_i^{-1}(f_i - F_i y_i)$$

Upon substituting a reduced system is obtained :

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

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

$$\begin{pmatrix} I & S_1^{-1} C_{12} & \cdots & S_1^{-1} C_{1N} \\ S_2^{-1} C_{21} & I & \cdots & S_2^{-1} C_{2N} \\ \vdots & & \ddots & \vdots \\ S_N^{-1} C_{N1} & \cdots & S_N^{-1} C_{NN-1} & I \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} \hat{g}_1 \\ \vdots \\ \hat{g}_N \end{pmatrix} \tag{4}$$

A solution method involves four steps :

- obtain the rhs of the reduced system.

$$\hat{g}_i = S_i^{-1} g_i - E_i B_i^{-1} f_i$$

- compute the LU decomposition of the local Schur complement matrix S_i.
    - a LU decomposition of A_i gives the LU decomposition of S_i.

$$A_i = \begin{pmatrix} B_i & F_i \\ E_i & S_i + E_i B_i^{-1} F_i \end{pmatrix} = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix}$$

$$\Leftrightarrow S_i = L_{S_i} U_{S_i}$$

- solve the reduced system.
- back-substitute to obtain the other unknowns (fully parallel step).

Schur Decomposition
Some difficulties

- In most real problem, the Jacobian is ill-conditioned.
  $\Rightarrow$ Need to use Preconditioned Schur Complement.

Example (V10Injection problem)

$$10^{+10} \leq cond \leq 10^{+16} \tag{5}$$

- The Schur complement matrix S is not built (high computational cost and time dependence).
  $\Rightarrow$ No direct solvers.
  $\Rightarrow$ Krylov solver.
  - if the Jacobian matrix freezes during some steps.
  $\Rightarrow$ Reuse the Krylov projection space.

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

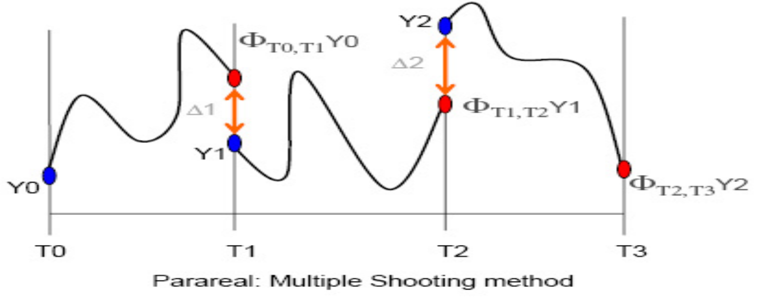
Schur Decomposition
Some results

#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

- With 3 processors, the speed-up is higher than using the parallelisation ”across the method”.
- It is not limited to 3 processors.
- The speed-up is supra-linear in this test case.
- These promising results can easily be applied to bigger problems.

Outline

Parareal Method : $\frac{\partial y}{\partial t} = f(t, y(t)), \quad t \in [T_0, T_f], \; y(T^0) = y_0.$



- Split the time interval into $S^i = [T_i, T_{i+1}]$ with $T_{N_{ts}} = T_f$.
- Then we solve in parallel

$$\frac{\partial y_k^i}{\partial t} = f(y_k^i, t), \quad t \in S^i, \; y_k^i(T^i) = Y_k^i. \tag{6}$$

- Finally the jumps Δ_k^i are then corrected.

$$\frac{\partial c_k}{\partial t} = f_y(t, y_k)c_k \text{ with } c_k(t_0) = 0 \text{ and } c_k(t^{i+}) = c_k(t^{i-}) + \Delta_k^i \tag{7}$$

S1 Init IBV with a coarse tolerance time integrator.(sequential).

$$y(T^{-}_{i,last}) \quad \rightarrow \quad y(T^{+}_{i+1,first})$$

S2 until convergence do

S2.1 Parallel solve of the ODEs system with a time integrator fine tolerance on subdomains $[T^{+}_{i,j}, T^{-}_{i,j+1}]$, $j = 1, \ldots, m$

S2.2 Algorithm 1 : *Sequential* correction process (Gauss-Seidel scheme)

$$c_i(T^{-}_{i,last}) + \Delta_{i,last} \quad \rightarrow \quad c_i(T^{+}_{i+1,first})$$

S2.2b Algorithm 2 : Parallel correction process (Jacobi scheme)

$$\Delta_{i,last} \text{ not sent} \rightarrow \tilde{c}_i(T^{-}_{i,last})$$

$$y_i(T^{-}_{i,last}) + \tilde{c}_i(T^{-}_{i,last}) \quad \rightarrow \quad y_i(T^{+}_{i+1,first})$$

Parallelism in Parareal : Algorithm 1 and Algorithm 2

S1 Init IBV with a coarse tolerance time integrator.(sequential).

$$y(T^-_{i,last}) \quad \rightarrow \quad y(T^+_{i+1,first})$$

S2 until convergence do

S2.1 Parallel solve of the ODEs system with a time integrator fine tolerance on subdomains $[T^+_{i,j}, T^-_{i,j+1}]$, $j = 1, \ldots, m$

S2.2 Algorithm 1 : *Sequential* correction process (Gauss-Seidel scheme)

$$c_i(T^-_{i,last}) + \Delta_{i,last} \quad \rightarrow \quad c_i(T^+_{i+1,first})$$

S2.2b Algorithm 2 : Parallel correction process (Jacobi scheme)

$$\Delta_{i,last} \text{ not sent} \rightarrow \tilde{c}_i(T^-_{i,last})$$

$$y_i(T^-_{i,last}) + \tilde{c}_i(T^-_{i,last}) \quad \rightarrow \quad y_i(T^+_{i+1,first})$$

Beam problem : Stiff system of ODEs of dimension $2n$

An anchored beam is governed by

$$z'' = C(z)v(z) + D(z)u(z, z')$$

where C is a tridiagonal matrix and D a bidiagonal matrix with lower and upper entries and with :

$$v_l = n^4(z_{l-1} - 2z_l + z_{l+1}) + n^2(cos(z_l)F_v - sin(z_l)F_u)$$

$$u = C^{-1}(Dv + z'^2)$$

$$F_u = -\phi(t),\ F_v = \phi(t),\ \phi(t) = \begin{cases} 1.5sin^2(t), 0 \leq t \leq \pi \\ 0, t \geq \pi \end{cases}$$

The problem is rewriten as a 1-order ODE

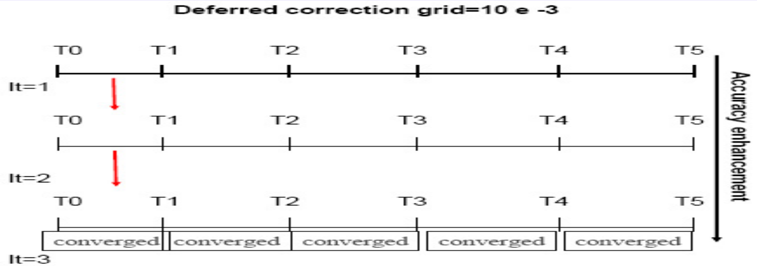
$$\begin{pmatrix} z \\ w \end{pmatrix}' = \begin{pmatrix} w \\ f(t, z, w) \end{pmatrix}$$

Results on Beam :

Beam problem with 200×2 unknowns				
elapse time (s)	# processors			
	2	4	8	16
Initializing $rtol = 10^{-2}$	1394	1366	1364	1366
Fine grid $rtol = 10^{-5}$	2526	1080	587	312
correction $rtol = 10^{-3}$	11260	4265	2184	1149
Total	15180	6711	4135	2827

TAB.: times and Parallel *efficiency* of Algorithm 2 on the beam problem, with $n = 200 \times 2$ unknowns to perform 3 parareal iterates.

- Relaxation communication on the correction step can enhance the parallelism but are still too high time comsuming.
- the correction problem is sensitive to the Jacobian linearizing and can blow-up if not sufficient care is taken on the size of subdomains.



- improve the accuracy of the time integration
- spectral convergence property : M. Minion Appl. Numer. Math. 48 (2004), no. 3-4, 369–387. Semi-implicit projection methods for incompressible flow based on spectral deferred corrections. Workshop on Innovative Time Integrators for PDEs.
- Sequential iterative process : we propose a parallel implementation to combine deferred correction method with time domain decomposition.

1. Compute an approximation y^0.
2. Iterate until convergence
   1. Compute an approximation δ^0_m of the defect $\delta(t_m) = y(t_m) - y^0_m$

      $$\delta^0_{m+1} = \delta^0_m + \int_{t_m}^{t_{m+1}} \left(f(\tau, y^0(\tau) + \delta(\tau)) - q^0(\tau)\right) \partial\tau - y^0_{m+1} + y^0_m + \int_{t_m}^{t_{m+1}} q^0(\tau)\partial\tau$$

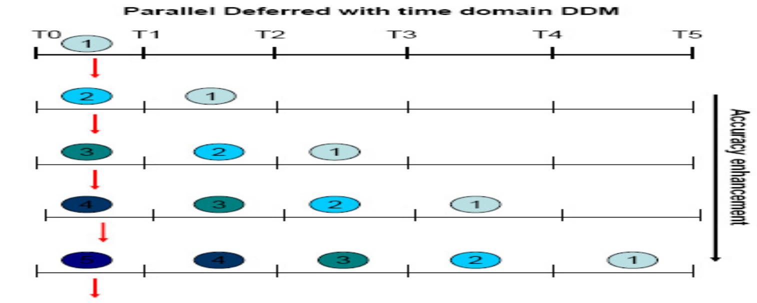
      where q^0_m a polynomial of degree k which interpolate $f(t_i, y^0(t_i))$ for $i \in [m - k/2, m + k/2]$ and $\int_{t_m}^{t_{m+1}} q^0(\tau)$ exactly solved by quadrature formula.
   2. Update the solution $y^1_m = y^0_m + \delta^0_m$.

1. Compute an approximation y^0.

$\Rightarrow$ Send the solution $y^0(t_{last-k} : t_{last})$ to next processor.

2. Iterate until convergence
   1. Compute δ^0_m

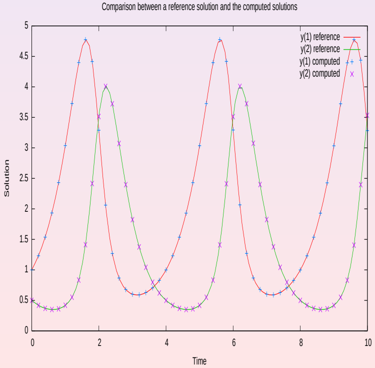
   $\Rightarrow$ Send $\delta^0(t_{last-k} : t_{last})$ to the next processor and $\delta^0(t_{begin} : t_{begin+k})$ to the previous one.
   2. Update the solution $y^1_m = y^0_m + \delta^0_m$.

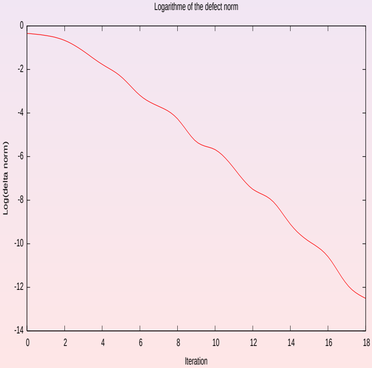


First Results on the DC

Sequential convergence study on a Lotka-Volterra problem (prey-predator problem)

$$\begin{cases} \dot{y}_1 = \mu_1 y_1 + \mu_2 y_1 y_2 \\ \dot{y}_2 = \mu_2 y_2 + \mu_3 y_1 y_2 \end{cases}$$





Outline

Conclusions

- The framework for Schur Decomposition applied to stiff ODE/ADE systems is set.
- Introduce an automatic graph framework to deal with complex systems.
- A combined time DDM and spectral deferred correction has been proposed... (parallel code on going)
- Future works :
  - Compare this framework with the existing data partitioning algorithms implemented in Sundials.
  - Introduce adaptivity in the partitioning when there are topological changes in the system during the integration.
  - Validation in progress on bigger problems.

Conclusions

- The framework for Schur Decomposition applied to stiff ODE/ADE systems is set.
- Introduce an automatic graph framework to deal with complex systems.
- A combined time DDM and spectral deferred correction has been proposed... (parallel code on going)
- Future works :
  - Compare this framework with the existing data partitioning algorithms implemented in Sundials.
  - Introduce adaptivity in the partitioning when there are topological changes in the system during the integration.
  - Validation in progress on bigger problems.