Parallel Computational Fluid Dynamics - Parallel Computing and Its Applications
*J.H. Kwon, A. Ecer, J. Periaux, N. Satofuka and P. Fox (Editors)*
2007 Elsevier B.V.


# Parallel Deferred Correction method for CFD Problems

D. Guibert [a] and D. Tromeur-Dervout[a, b] *

[a]ICJ UMR5208 Université Lyon 1 - CNRS , Centre de Développement du Calcul Scientifique Parallèle Université Lyon 1

[b]IRISA/SAGE INRIA Rennes

## 1. Motivation of Time Decomposition in the Parallel CFD context

Time domain decomposition methods can be a complementary approach to improve parallelism on the forthcoming parallel architectures with thousand of processors. Indeed, for a given modelled problem, - e.g. through a partial differential equation with a prescribed discretization- the multiplication of the processors leads to a decrease of the computing load allocated to each processors, then the ratio of the computing time with the communication time decreases, leading to less efficiency. Nevertheless , for the evolution problems, the difficulty comes from the sequential nature of the computing. Previous time steps are required to compute the current time step. This last constraint occurs in all time integrators used in computational fluid dynamics of unsteady problems.

During these last years, some attempts have been proposed to develop parallelism in time solvers. Such algorithms as Parareal [6] or Pita [2] algorithms are multiple shooting type methods and are very sensitive to the Jabobian linearization for the correction stage on stiff non-linear problems as shown in [4].

Moreover, this correction step is an algorithm sequential and often much more time consuming to solve for the time integrator than the initial problem.

In this paper we investigate a new solution to obtain a parallelism in time with a two level algorithm that combines pipelined iterations in time and deferred correction iterations to increase the accuracy of the solution. In order to focus on the benefit of our algorithm we approach CFD problems as an ODE (eventually DAE) problem. From the numerical point of view the development of time domain decomposition methods for stiff ODE problems allows to focus on the time domain decomposition without perturbation coming from the space decomposition. In other side, classical advantages of the space decomposition on the granularity are not available. Indeed, from the engineering point of view in the design with ODE/DAE, the efficiency of the algorithm is more focus on the saving time in the day to day practice.

*This work was funded by the thema "mathématiques" of the Région Rhône-Alpes thru the project: "Développement de méthodologies mathématiques pour le calcul scientifique sur grille"

## 2. ODE approach for CFD problem

The traditional approach to solve CFD problems starts from the partial differential equation, which is discretized in space with a given accuracy and finally a time discretization with a first order scheme or a second order time scheme is performed to solve problems ( usually implicitly due to Courant-Friedrich-Lax condition constraint).

Let consider the 2D Navier-Stokes equations in a Stream function - vorticity formulation:

$$\begin{cases} \omega_t + \psi_x \omega_y - \psi_y \omega_x - \frac{1}{Re}\Delta\omega = 0 \\ \Delta\psi = -\omega \\ \psi = 0, \psi_n = v(t) \text{ on } \partial\Omega \end{cases} \tag{1}$$

then applying a backward ( forward Euler or Runge-Kutta are also possible) time discretisation, we obtain that follows:

$$\begin{cases} \omega^{n+1} + \Delta t(\psi_x^n \omega_y^{n+1} - \psi_y^n \omega_x^{n+1}) - \frac{\Delta t}{Re}\Delta\omega^{n+1} = \omega^n \\ \Delta\psi^{n+1} = -\omega^{n+1} \end{cases} \tag{2}$$

The discretization in space of the discrete in time equations leads to solve a linearized system with an efficient preconditionned solver such as Schwarz-Krylov method or multigrid:

$$\{ \ A(\psi^n)\omega^{n+1} = f(\psi^n), \ \ \Delta\psi^{n+1} = -\omega^{n+1} \tag{3}$$

Let us consider an another approach that is the method of lines or ODE to solve CFD problems. Firstly, the Navier-Stokes problem is written in a Cauchy equation form:

$$\frac{d}{dt}\begin{pmatrix} \omega \\ \psi \end{pmatrix}(t) = f(t, \omega, \psi), \ \begin{pmatrix} \omega \\ \psi \end{pmatrix}(0) = \begin{pmatrix} \omega_0 \\ \psi_0 \end{pmatrix} \tag{4}$$

then the discretization in space of the $f(t, \omega, \psi)$ leads to a system of ODE :

$$\frac{d}{dt}\begin{pmatrix} \omega_{i,j} \\ \psi_{i,j} \end{pmatrix} = f(t, \omega_{i[\pm 1],j[\pm 1]}, \psi_{i[\pm 1],j[\pm 1]}) \tag{5}$$

This last equation can be solved by an available ODE solver such as SUNDIALS ( SUite of Nonlinear and DIfferential/ALgebraic equation Solvers (http://www.llnl.gov/casc/sundials/)) that contains the latest developments of classical ODE solvers as LSODA [8] or CVODE [1].

Figure 1 corresponds to the lid driven cavity results computed by the ODE method a Reynold number of 5000 and up to 10000. These results are obtained for the impulse lid driven with a velocity of 1. Let us notice that the convergence is sensitive to the singularities of the solution.

The advantages and drawbacks of the PDEs and ODE are summarized in Table 1

In conclusion the ODE approach put the computing pressure on the time step control while PDEs approach put the computing pressure on the preconditionned linear/nonlinear solver.

The high time consumming drawback for ODE explains why this approach is not commonly use in CFD. Nevertheless grid computing architecture could give a renewed interest to this ODE approach.

| | PDEs | ODE |
|---|---|---|
| 1 | Fast to solve one time step | slow to solve one time step<br>(control of time step) |
| 2 | Decoupling of variables | coupling of variables<br>(closest to incompressibility constraint) |
| 3 | computing complexity is reduced<br>with the decoupling | high computing complexity ()<br>control of error, whole system |
| 4 | B.C. hard to handle with<br>impact on the linear<br>solver (data structure/choice) | B.C. conditions easy<br>to implement |
| 5 | CFL condition even with implicit<br>solve (decoupling of variables) | CFL condition replaced<br>by the control on time step |
| 6 | Building of good preconditionner<br>is difficult | Adaptive time step allows<br>to pass stiffness |
| 7 | Adaptive time step hard<br>to handle | Adaptive time step is<br>a basic feature in ODE |

Table 1
Advantages and drawbacks of ODE and PDEs approaches for CFD problems

## 3. Spectral Deferred Correction Method

The spectral deferred correction method (SDC) [7] is a method to improve the accuracy of a time integrator iteratively in order to approximate solution of a functional equation with adding corrections to the defect.

Let us summarize the SDC method for the Cauchy equation:

$$y(t) = y_0 + \int_0^t f(\tau, y(\tau))d\tau \tag{6}$$

Let $y_m^0 \simeq y(t_m)$ an approximation of the searched solution at time $t_m$. Then consider the polynomial $q_0$ that interpolate the function $f(t, y)$ at P points $\tilde{t}_i$ i.e:

$$\exists! q_0 \in R_P[x], q_0(\tilde{t}_i) = f(y^0(\tilde{t}_i)) \tag{7}$$

with $\left\{\tilde{t}_0 < \tilde{t}_1 < ... < \tilde{t}_P\right\}$ are points around $t_m$. Then if we define the error between the approximate solution $y_m^0$ and the approximate solution using $q_0$:

$$E(y^0, t_m) = y_0 + \int_0^{t_m} q^0(\tau)d\tau - y_m^0 \tag{8}$$

Then we can write the error between the exact solution and the approximation $y_m^0$:

$$y(t_m) - y_m^0 = \int_0^{t_m} \left(f(\tau, y(\tau)) - q^0(\tau)\right) d\tau + E(y^0, t_m) \tag{9}$$

The defect $\delta(t_m) = y(t_m) - y_m^0$ satisfies:

$$\delta(t_{m-1}) = \delta(t_m) + \int_{t_m}^{t_{m+1}} \left(f(\tau, y^0(\tau) + \delta(\tau)) - q^0(\tau)\right) d\tau + E(y^0, t_{m+1}) - E(y^0, t_m) \tag{10}$$

For the spectral deferred correction method an approximation $\delta_m^0$ of $\delta(t_m)$ is computed considering for example a Forward Euler scheme :

$$\delta_{m+1}^0 = \delta_m^0 + \Delta t(f(t_{m+1}, y_{m+1}^0 + \delta_{m+1}^0) - f(t_{m+1}, y_{m+1}^0)) - y_{m+1}^0 + y_m^0 + \int_{t_m}^{t_{m+1}} q^0(\tau)d\tau$$

Once the defect computed , the approximation $y_m^0$ is updated

$$y_m^1 = y_m^0 + \delta_m^0. \tag{11}$$

Then a new $q_0$ is computed with using the value $y_m^1$ for a new iterate. This method is a sequential iterative process as follows: compute an approximation $y^0$, then iterate in parallel until convergence compute $\delta$ and update the solution $y$ by adding the correction term $\delta$.

We propose in the next section a parallel implementation to combine deferred correction method with time domain decomposition.

## 4. Parallel SDC: a time domain decomposition pipelined approach

The main idea to introduce parallelism in the SDC is to consider the time domain decomposition and to affect a set of time slices to each processors. Then we can pipeline the computation, each processor dealing a different iterate of the SDC.

**Algorithm 4.1** (Parallel Version). *1 spread the time slices between processors with an overlap $P/2$ slices and compute an approximation $y^0$ with a very low accuracy because this is a sequential stage. Each processor keeps only the time slices that it has to take in charge, and can start the pipelined as soon it has finished.*

*2 Iterate until convergence*

*2.1 prepare the non blocking receiving of the last $P/2$ delta value of the time interval take in account by the processor.*

*2.2 If the left neighbor processor exist wait until receiving the $P/2$ $\delta$ value from it in order to update the $y_m^i$ in the overlap of size $P/2$.*

*2.3 Compute $q_0$ from $y_m^i$ in order to compute $\delta_m^i$ When the $\delta$ values between $P/2+1$ and $P$ of time slices are computed send to left neighbor processor in order it will be able to start the next iterate (corresponding to the stage 2.1) .*

*2.4 Send the $P/2$ $\delta$ values corresponding to the overlap position of the right neighbor processor.*

*2.5 Update the solution $y_m^{i+1} = y_m^i + \delta_m^i$*

**Remark 4.1.** *The stage 1 initialization should not be parallelized in order to keep the time integrator history and avoid jump between the initial value inside the domain and the value in the overlap on the neighbor processor.*

**Remark 4.2.** *The communication stages [2.1] and [2.2] are needed to compute the $q_0$ polynomial that uses the behavior of the function $f(t, y)$ before and after the point $t_m$.*

| #proc | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| [1] (s) | 17.12 | 18.68 | 17.36 | 16.41 |
| [2.2] (s) | 1.35e-5 | 72.39 | 113 | 129 |
| [2.3] $f(t,y)$ eval (s) | 1.66e-2 | 7.25e-3 | 3.13e-3 | 1.126e-3 |
| [2.3] $q_0$ computing (s) | 5.99e-3 | 5.5e-3 | 5.90e-3 | 5.94e-3 |
| euler | 0.288 | 0.25 | 0.19 | 0.16 |
| Total(s) | 2062 | 1113 | 742 | 423 |
| Efficiency | 100% | 92.6% | 69.4% | 60.9% |
| Speed-up | 1 | 1.85 | 2.77 | 4.87 |
| Speed up Corrected | 1 | 1.98 | 3.27 | 7.01 |

Table 2
Time, Speed-up and Efficiency of the Time domain decomposition pipelined SDC

Table 2 gives the times to run the time domain decomposition pipelined SDC method on an ALTIX350 with IA64 1.5Ghz/6Mo processors and with numalink communication network. The lid driven parameters are a Reynolds number of 10 and a regular mesh of $20 \times 20$ points, the final time to be reach is T=5. The number of time slices has been set equal to 805. The number of SDC iteration is 25. The stage [1] correspond to the sequential first evaluation of the solution on the time slices (with an rtol of $10^{-1}$). The time [2.2] is the time of the pipelined to reach the last processor, results show that the ratio of this time [2.2] on the computation is increasing up to 1/4 for 8 processors case. The Speed-up corrected line gives the parallel capability of the parallel implementation when the pipeline is full.

It exhibits quite reasonable speed-up up to 8 processors for the present implementation.

**Remark 4.3.** *Some improvement of the presented implementation can be performed. firstly we implement the forward Euler to compute the $\delta_m^{i+1}$ that needs a Newton solving at each iteration with the Jacobian computed by a numerical differentiation. We also can use the sundials solver with blocking at first order the BDF. Secondly, all the value $f(\tilde{t}_m, y_m^i)$ are computed before and could be progressively computed in order to enhance the chance of parallelism. Thirdly, a better load balancing could be acheived with a cyclic distribution of the time slices among the processors.*

Figure 2 gives the convergence of the defect on the time interval for different iteration numbers (1,5,9,13,17). Symbols represent the runs on 1 processor (x), 2 processors (o) , 8 processors (□). Firstly, we remark that the number of processors does not impact the convergence properties of the method. Secondly, the convergence behavior increases for the last times due to the steady state of the solution. Nevertheless, a good accuracy is obtained on the first times in few SDC iterates.

## 5. conclusions

In this work we have investigated the numerical solution of 2D Navier-stokes with ODE solvers in the perspective of the time domain decomposition which can be an issue for the

grid computing.

we proposed to combine time domain decomposition and spectral deferred correction in order to have a two level algorithm that can be pipelined. The first results proved some parallel efficiency with a such approach.

Nevertheless, the arithmetical complexity of the ODE solver for the lid driven problem consider here, is too much costly. Some scheme such as the $C(p,q,j)$ [3] schemes can decouple the system variables in order to reduce the complexity.

## REFERENCES


1. S. D. Cohen and A. C. Hindmarsh. *CVODE, a Stiff/Nonstiff ODE Solver in C.* Computers in Physics, 10(2):138-143, 1996.
2. C. Farhat and M. Chandesris. *Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications.* Int. J. Numer. Meth. Engng., **58**(9):1397–1434, 2003.
3. M. Garbey and D. Tromeur-Dervout,*A parallel adaptive coupling algorithm for systems of differential equations*, J. Comput. Phys.,161(2):401 427,2000
4. D. Guibert and D. Tromeur-Dervout, *Parallel Adaptive Time Domain Decomposition for Stiff Systems of ODE/DAE*, Computer and Structure, 2006 to appear.
5. E. Hairer, S. P. Norsett and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer Series in Computational Mathematics, Springer, Berlin, 1991.
6. J.-L. Lions, Y. Maday, and G. Turinici. *Résolution d'EDP par un schéma en temps "pararéel"*, C.R.A.S. Sér. I Math., **332**(7):661 668, 2000.
7. Minion, Michael L., *Semi-implicit spectral deferred correction methods for ordinary differential equations*, Commun. Math. Sci.,1(3):471–500,2003.
8. L. R. Petzold, *A Description of DASSL: A Differential and Algebraic System Solver* SAND82-8637 (September 1982).

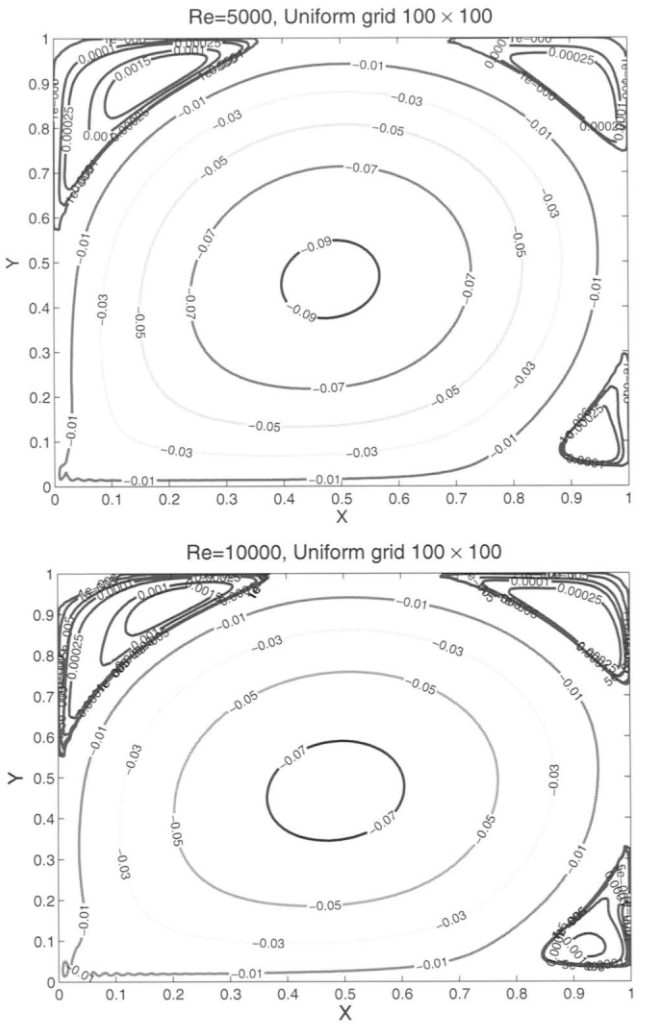


Figure 1. Impulse lid driven cavity results for a Reynold number of 5000 (top) and 10000 (bottom) with the ODE approach

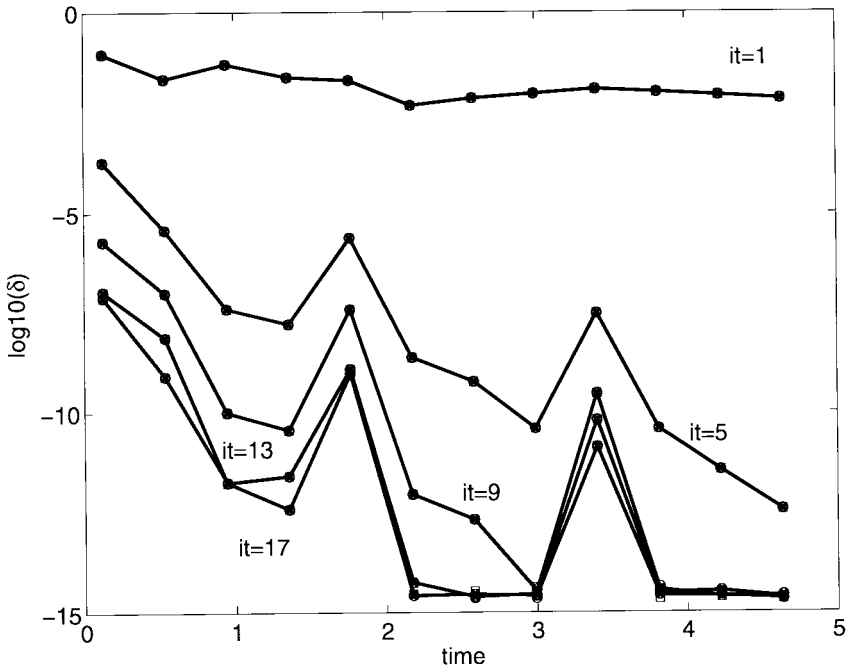


Figure 2. Convergence on the time interval of the defect $\delta$ for different iteration numbers (1,5,9,13,17). Symbols represent the runs on 1 processor (x), 2 processors (o) , 8 processors (□)