In this paper, we propose a numerical strategy to speed up the implicit solution of unsteady nonlinear problems arising from fluid dynamics. This strategy consists in a partial update of a domain decomposition preconditioner used in the Newton-Krylov method that solves the nonlinear problem of each time step. The underlying principle of the proposed method is that, usually, there is only slight changes between two consecutive Jacobian matrices. Consequently, it is possible to use the same preconditioner for few Newton iterations, or, even better, to partially update it. We propose to add some processes dedicated to the asynchronous update of the subdomains parts of the preconditioner. Numerical results for the lid-driven cavity are provided, they show that this addition of processes can speed up the computation in a super linear way.

Abstract

In this paper, we propose a numerical strategy to speed up the implicit solution of unsteady nonlinear problems arising from fluid dynamics. This strategy consists in a partial update of a domain decomposition preconditioner used in the Newton-Krylov method that solves the nonlinear problem of each time step. The underlying principle of the proposed method is that, usually, there is only slight changes between two consecutive Jacobian matrices. Consequently, it is possible to use the same preconditioner for few Newton iterations, or, even better, to partially update it. We propose to add some processes dedicated to the asynchronous update of the subdomains parts of the preconditioner. Numerical results for the lid-driven cavity are provided, they show that this addition of processes can speed up the computation in a super linear way.

Introduction

The spatial discretization of many CFD problems produces ordinary differential equations of the form Eq. (1) for a given initial condition \( x(0) = x_0 \) and suitable boundary conditions.

\[ \dot{x} + L(x(t)) = g(t) \]  

(1)

In Eq. (1), \( L(x) \) is a nonlinear discrete operator from \( \mathbb{R}^n \) to \( \mathbb{R}^n \) representing a spatial approximation of a parabolic boundary values, and \( g \) is a source term. Then, the time discretization of Eq. (1) by an implicit method leads to solving a system of nonlinear equations of the form \( F(x) = 0 \) at each time step. For example, if we consider the implicit Euler method with a fixed time step, then \( F(x^i) = x^i + \Delta t L(x^i) - x^{i-1} - \Delta t g(t_i) \) where \( x^i \approx x(t_i) \) and \( t_i = t_0 + i \Delta t \). One of the most common methods to solve such nonlinear systems is the Newton-Krylov method (see [1] and references therein) that involves the solution of linear systems of the form

\[ J(x_k) \Delta x_k = -F(x_k) \]  

(2)

where \( J(x_k) \in \mathbb{R}^{n \times n} \) is the Jacobian matrix of \( F \) at \( x_k \), or an approximation of it. The condition number of the matrix \( J(x_k) \) can be very large, hence, a good preconditioner is required. The Restricted Additive Schwarz preconditioner
(RAS) [2] has a good capability to precondition the Newton method [3]. Solving the linear systems is generally the most time consuming part of the codes, even if there are usually only slight changes between two consecutive linear systems. In order to save computations, the preconditioning matrix could be frozen during few Newton’s iterations, assuming that a preconditioner may be efficient even if it was computed for a previous Jacobian matrix. We propose to partially update the restricted additive Schwarz preconditioner. The RAS preconditioner of the linear system Eq. (2) can be written as Eq. (3) where \( N \) is the number of subdomains, \( J_i \) is the local part of the Jacobian matrix associated to the \( i \)th subdomain, and \( R_i \) (resp. \( \tilde{R}_i \)) is the restriction operator to the \( i \)th subdomain with (resp. without) the overlap between subdomains.

\[
M_{\text{RAS}}^{-1} = \sum_{i=1}^{N} \tilde{R}_i^T J_i(x_k)^{-1} R_i
\]

In the first section we detail algorithms for a frozen RAS preconditioner and its partial update. In the second section, we propose a parallel implementation of the algorithm, where few processors are added to the classical implementation, these processors will be in charge of recomputing the local part of the preconditioner in a client-server approach. The third section is devoted to numerical experiments on the lid-driven cavity problem.

1. Newton-Krylov method with a frozen RAS preconditioner

The Newton-Krylov-Schwarz (NKS) [4, 5] is often used to solve nonlinear problems arising in CFD. The application of RAS the preconditioner requires the solution of subdomain’s linear systems. When the Jacobian matrix slightly changes from one Newton iteration to another, it could be relevant to compute the LU factorizations of the subdomain’s part of the Jacobian matrix. Hence, the same preconditioner can be applied during few Newton’s iterations. Algorithm 1 puts this idea into practice: the preconditioning matrix is updated (i.e. the local LU factorizations are computed) when the number of Krylov iterations of the previous linear system exceeds \( K_{\text{max}} \).

**Algorithm 1** Time stepper with a frozen RAS preconditioner

**Require:** initial guess \( x \), restarting criterion \( K_{\text{max}} \), \( k = 0 \)

**Require:** the partitioning of the unknowns into \( N \) subsets

1: for each time step do
2:     // Newton iterations:
3:     repeat
4:         if \( k > K_{\text{max}} \) then
5:             update \( P = M_{\text{RAS}}^{-1} = \sum_{i=1}^{N} \tilde{R}_i^T J_i(x)^{-1} R_i \) (LU factorization of each \( J_i \))
6:         end if
7:     solve \( PJ_i(x)\Delta x = -PF(x) \) with a Krylov method
8:     \( k \leftarrow \) number of Krylov iterations
9: x \leftarrow x + \Delta x
10: until convergence
11: end for

In Algorithm 1, all the local LU factorizations are computed simultaneously. In practice, there may be highly nonlinear phenomenas localized on certain subdomains. In that case, only few blocks of the Jacobian matrix significantly changes from a Newton iteration to another. Then, the frequency with which the LU factorizations of the local Jacobian \( J_i \) may be updated varies from one subdomain to another.

In Algorithm 2 (step 11), \( I \) is the index set of the subdomains for which the LU factorization will be computed at the next iteration. This index set \( I \) can be set a priori if the nonlinear behavior of the problem is known, or after the solution of the linear system using some numerical criteria. Let us note that the preconditioner can be written as in Eq. (4), that is to say that each LU factorization corresponds to the Jacobian matrix evaluated at \( x_{k_i}^i \) corresponding to the \( k_i \)th Newton iteration of the time step \( t_i \).

\[
P = \sum_{i=1}^{N} \tilde{R}_i^T J_i(x_{k_i}^i)^{-1} R_i
\]
Algorithm 2 Partial update of the RAS preconditioner

Require: initial guess $x$, $I = \{1, 2, \ldots, N\}$

1: for each time step $t$ do
2:     $k \leftarrow 1$ // Newton iterations:
3:     repeat
4:         for each $i \in I$ do
5:             $k_i \leftarrow k, t_i \leftarrow t$
6:             compute the LU factorization of $J_i(x_{k_i}^t)$
7:         end for
8:     solve $PJ(x_{k}^t)\Delta x = -PF(x_{k}^t)$ with $P = \sum_{i=1}^{N} \tilde{R}_i^T J_i(x_{k}^t)^{-1} R_i$.
9:     $x_{k+1}^t \leftarrow x_{k}^t + \Delta x, k \leftarrow k + 1$
10:    until convergence
11: end for

The sequential implementation of Algorithm 2 is straightforward. If implemented on a distributed memory computer, this algorithm will not be useful: if some processors have to compute the LU factorizations, all processors should compute its own LU factorization instead of waiting. To circumvent this difficulty, in the next section we propose to dedicate additional processes to the LU factorizations.

2. Asynchronous implementation

The method presented in the previous section does not seem optimal in terms of parallel computing because the load is not balanced: some of the processors will have to wait while the other ones compute the LU factorizations. In order to avoid idle time, asynchronous solvers have been studied for linear and nonlinear problems [6, 7, 8, 9]. In the following we propose an asynchronous implementation of the preconditioner given in Eq. (4). The advantage of this method is that the preconditioner has an effect on the convergence speed of the Krylov method, but not on the solution itself.

2.1. The algorithm

We can define two kinds of tasks: the LU factorization, and the Newton’s iterations within the time-stepper. Then, in order to solve the physical problem, one must assign some processes to each task. Furthermore, it is possible to compute the LU factorizations asynchronously relatively to time-stepper iterations. Algorithm 3 describes how to implement the method in a client-server approach. The client processes are those which are in charge of a subdomain, while the server ones are devoted to the computation of LU factorizations. The client processes must be able to continue the computation between the send of the local part of the Jacobian matrix and the reception of the factorized matrix. The reception can be done between two Newton iterations or even between two Krylov iterations if this one allows variable preconditioners [10, 11].

Let also notice that a restart can also be performed for robustness reasons. Typically, when a given maximum number of Krylov iterations $K_{max}$ is reached, then all the client processes compute the preconditioner simultaneously.

2.2. Parallel implementation

We now discuss the MPI implementation of the Algorithm 3 for a computer cluster: computational nodes in which few cores share memory, linked together by a network connection. The main difficulty to achieve an efficient MPI implementation of the algorithm is to choose the mapping of the tasks to the cores. The following points should be taken into account:
Algorithm 3 Asynchronous update of the preconditioner

Require: initial guess $x$
1: if client process associated to a physical subdomain then
2:    for each time step do
3:        // Newton iterations:
4:        repeat
5:            if $k > K_{\text{max}}$ then
6:                $P = \sum_i \tilde{R}_i^T J_i(x)^{-1} R_i$
7:            end if
8:            Solve $PJ(x) \Delta x = -PF(x)$ by a Krylov method
9:            $x \leftarrow x + \Delta x$
10:         $k \leftarrow$ number of Krylov iterations
11:        If needed, send a request to the server
12:        until convergence
13:    end for
14: else
15:    // Server process performing LU factorizations
16:    repeat
17:        Receive the request
18:        Perform the LU factorization
19:        Send back the factorized matrix
20:    until End of simulation
21: end if

• The cores that compute LU factorizations and its clients asynchronously exchange large messages (matrices, and factorized matrices).
• The processes associated to a physical subdomains will frequently exchange small messages with those in charge of the neighboring subdomains.

Another possibility is to compute the preconditioner on co-processors: a decent speedup can be obtained, but the computational power of the co-processor may be underused because of the data transfers. Generally speaking, one should avoid the exchange of factorized matrix through the network, then a MPI library performing efficient intranode communications is required. Consequently, tasks that compute the LU factorization and those that solve the subdomain’s problems should share memory. Then, one has to set the number of cores per node which are devoted to the LU factorizations. Let us remark that depending on the physical problem, the mapping of the subdomain to cores must be adapted in order to balance the computations. Thus, one should scatter the processes that require a lot of LU factorizations, that is to say those in charge of highly nonlinear subdomain’s problem. One the other hand, this could lead to the scattering of neighbouring subdomain’s problem that communicate frequently.

3. Numerical tests

The numerical experiments are performed for the lid-driven cavity problem on the unit square. The PETSc library [12] was used for the implementation.

\[
\begin{align*}
-\Delta (u) - \nabla_y (\omega) &= 0 \\
-\Delta (v) + \nabla_x (\omega) &= 0 \\
\dot{\omega} - \Delta (\omega) + \nabla \cdot ([u \times \omega, v \times \omega]) &= 0
\end{align*}
\] (5)

In Eq. (5), $u$ and $v$ are the two components of the velocity field, and $\omega = -\nabla_y u + \nabla_x v$ is the vorticity. The space discretization is performed on a regular grid with a five-point stencil and the time discretization is a backward Euler scheme, the equation is treated as an ordinary differential equation (the terms $\dot{u}$, $\dot{v}$ are added to the two first equations).
The lid-velocity \( u(x, 0) \) is a nonzero constant, the other boundary conditions satisfy \( u = v = 0 \). The computer cluster used for the numerical experiment is an SGI Altix XE 1300, with two Intel Xeon 5650 per node.

3.1. Speedup resulting from the externalization of the LU factorizations

In the following results, \( NK + LRAS \) refers to the Newton-Krylov method with a lagged RAS preconditioner (i.e. Algorithm 1), while \( NK + extLU \) designates method proposed in Algorithm 3 where the LU factorizations are computed by processes external to the Newton-Krylov method.

The first numerical test intends to output the effect of the addition of process dedicated to the LU factorizations. Therefore, we compare the computational times and for:

- The lid-driven cavity problem discretized on \( 900 \times 900 \) grid. The domain is decomposed in 100 subdomains, and the solver is a Newton-Krylov-Schwarz methods with an overlap of one cell. The RAS preconditioner is recomputed when the number of Krylov iterations reach \( K_{max} \) for a linear system.
- The same problem, with 20 additional cores dedicated to the LU factorizations.

Then, we compare the solution of the same problem using 100 or 120 cores. If we assume that the Newton-Krylov method has a linear strong scaling for small changes of the number of processes, consequently, it is relevant to assign additional processes to the LU factorizations if it leads to super linear strong scaling. Therefore, the proposed method will be said competitive if the addition of the 20 processes dedicated to the factorizations divides the walltime by a factor of at least 1.2. In the following this factor is called speedup.

Table 1 shows that:

- The more often the whole preconditioner is restarted, the less the external LU factorization are useful, and as a result the best speedups are obtained for large values of \( K_{max} \).
- The proposed algorithm \( NK + extLU \) is more robust: the walltime is less sensitive to the restarting criterion.
- For all the tests presented here, the speedup is above 1.2, even for the \( K_{max} = 100 \) (i.e. the lowest walltime for \( NK + LRAS \)).

<table>
<thead>
<tr>
<th>Restarting criterion ( K_{max} )</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>( NK + extLU )</td>
<td>172.2</td>
<td>168.0</td>
<td>169.5</td>
<td>165.5</td>
<td>175.2</td>
</tr>
<tr>
<td>( NK + LRAS )</td>
<td>221.1</td>
<td>215.3</td>
<td>221.9</td>
<td>227.4</td>
<td>298.9</td>
</tr>
<tr>
<td>speedup</td>
<td>1.284</td>
<td>1.281</td>
<td>1.309</td>
<td>1.374</td>
<td>1.706</td>
</tr>
</tbody>
</table>

The lid velocity on the top is 100. \( NK + extLU \) corresponds to Algorithm 3, \( NK + LRAS \) corresponds to Algorithm 2.

3.2. Increase of the load per process

For a fixed number of processors, a decrease of the efficiency of the proposed algorithm is expected when the problem size is increased because the computational time of the LU factorizations increases faster than the computational time of the Krylov iterations.

Table 2 presents the speedups obtained for the lid-driven cavity problem for different sizes of mesh and different restarting parameters \( K_{max} \). These speedups are super linear (greater than 1.2) but they decrease when the load per processor is increased.

It should be added that the proposed method (i.e. \( NK + extLU \)) is much less sensitive to the restart parameter \( K_{max} \) than the Newton-Krylov with a lagged preconditioner.

4. Conclusions

We presented a method that consists in adding processes devoted to the computation of the preconditioner involved in the Newton-Krylov solver. The utilization of domain decomposition preconditioners allows us to update only
Table 2. Speedup for the lid-driven cavity problem

<table>
<thead>
<tr>
<th>Size of the mesh</th>
<th>$k_{max}$</th>
<th>Wall time (s) for $NK + \text{LRAS}$</th>
<th>Wall time (s) for $NK + \text{extLU}$</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>$600 \times 600$</td>
<td>50</td>
<td>100.8</td>
<td>77.9</td>
<td>1.29</td>
</tr>
<tr>
<td>$600 \times 600$</td>
<td>100</td>
<td>145.4</td>
<td>90.6</td>
<td>1.61</td>
</tr>
<tr>
<td>$600 \times 600$</td>
<td>200</td>
<td>137.0</td>
<td>77.4</td>
<td>1.77</td>
</tr>
<tr>
<td>$600 \times 600$</td>
<td>300</td>
<td>159.1</td>
<td>79.5</td>
<td>2.00</td>
</tr>
<tr>
<td>$900 \times 900$</td>
<td>50</td>
<td>245.1</td>
<td>152.4</td>
<td>1.61</td>
</tr>
<tr>
<td>$900 \times 900$</td>
<td>100</td>
<td>236.6</td>
<td>155.0</td>
<td>1.53</td>
</tr>
<tr>
<td>$900 \times 900$</td>
<td>200</td>
<td>236.3</td>
<td>152.3</td>
<td>1.55</td>
</tr>
<tr>
<td>$900 \times 900$</td>
<td>300</td>
<td>264.9</td>
<td>148.5</td>
<td>1.78</td>
</tr>
<tr>
<td>$1200 \times 1200$</td>
<td>50</td>
<td>392.0</td>
<td>268.3</td>
<td>1.46</td>
</tr>
<tr>
<td>$1200 \times 1200$</td>
<td>100</td>
<td>387.3</td>
<td>320.1</td>
<td>1.21</td>
</tr>
<tr>
<td>$1200 \times 1200$</td>
<td>200</td>
<td>385.9</td>
<td>277.6</td>
<td>1.39</td>
</tr>
<tr>
<td>$1200 \times 1200$</td>
<td>300</td>
<td>384.5</td>
<td>269.2</td>
<td>1.43</td>
</tr>
<tr>
<td>$1500 \times 1500$</td>
<td>50</td>
<td>589.7</td>
<td>439.5</td>
<td>1.34</td>
</tr>
<tr>
<td>$1500 \times 1500$</td>
<td>100</td>
<td>584.7</td>
<td>435.6</td>
<td>1.34</td>
</tr>
<tr>
<td>$1500 \times 1500$</td>
<td>200</td>
<td>582.5</td>
<td>440.2</td>
<td>1.32</td>
</tr>
<tr>
<td>$1500 \times 1500$</td>
<td>300</td>
<td>590.6</td>
<td>445.6</td>
<td>1.33</td>
</tr>
</tbody>
</table>

Boundary condition: $u(x, 0) = 400$; run on 100 cores for $NK + \text{LRAS}$, and 120 cores for $NK + \text{extLU}$

certain subdomains parts of the preconditioner. Furthermore, this partial update can be computed asynchronously, that is to say that the time-stepper computations are not blocked. Finally, numerical results showed that super linear speedups can be obtained by adding processes dedicated to the LU factorizations because it allows to improve the load balance between the subdomains. A straightforward extension of this work is to compute ILU factorizations on additional processes if the local solution is computed by a Krylov method (instead of LU factorization).

Acknowledgements

This work has been supported by the French National Agency of Research (project ANR-MN2012-H2MNO4), and the ARC6 project of the région Rhône-Alpes. Authors also thank the Center for the Development of Parallel Scientific Computing (CDCSP) of the University of Lyon 1 for providing us with computing resources.

References