Available online at www.sciencedirect.com

ScienceDirect

Advances in Engineering Software 38 (2007) 301–311

ADVANCES IN ENGINEERING SOFTWARE

www.elsevier.com/locate/advengsoft

POD acceleration of fully implicit solver for unsteady nonlinear flows and its application on grid architecture

D. Tromeur-Dervout [a,*,1], Y. Vassilevski [b,2]

[a] Center for the Development of Parallel Scientific Computing, UMR5208 Université Lyon1-CNRS, Institut Camille Jordan, 15Bd Latarjet, 69622 Villeurbanne, France

[b] Institute of Numerical Mathematics, 8, Gubkina Street, 119991 Moscow, Russia

Received 12 September 2005; accepted 3 August 2006
Available online 9 October 2006

Abstract

A method for the acceleration of a fully implicit solution of nonlinear unsteady boundary value problem is presented. The principle of acceleration is for provide to the inexact Newton backtracking method a better initial guess, for the current time step, than the conventional choice from the previous time step. This initial guess is built on the reduced model obtained by a proper orthogonal decomposition of solutions at the previous time steps. This approach is appealing to GRID computing: spare processors may help to improve the numerical efficiency and to manage the computing in a reliable way.
© 2006 Elsevier Ltd. All rights reserved.

MSC: 65Y05; 65Y10; 65B99; 68H14; 35K55; 76

Keywords: Parallel computation; Acceleration of convergence; Algorithms for specific class of architecture; Distributed system; Nonlinear PDE of parabolic type; Fluid mechanics

1. Introduction

In the field of high performance computing architecture characteristics always impact on the development of computational methods. Novel numerical methodology may be attributed to the computer architecture and the associated software constraints/capabilities. The development of communication network performances and infrastructures such as the cost of computing resources, allows us to define computing architectures that gather several remotely located distributed computers to solve one application. Metacomputing and GRID computing refer to these computing architectures. Metacomputing, as defined by Smarr [20], uses a few stable high performance computers with a secured environment through dedicated or nondedicated communication network. GRID computing uses computing resources that are shared by several users, perhaps subject to hardware failures, through a nondedicated communication network.

Metacomputing has been implemented by many projects of which GLOBUS is the most widely known [9]. Experiments with large configurations and real applications have shown that the latency of wide area networks is prohibitively high and that substantial bandwidth is rarely achievable [17]. As a result some have concluded that metacomputing does not make sense. However, with the introduction of clusters of fast nodes, varying latencies and bandwidth have become typical features of most modern hardware architectures. Metacomputing can, therefore, serve as a test bed for the development of algorithms for

* Corresponding author. Tel.: +33 4 72 43 13 56; fax: +33 4 72 43 11 45.
E-mail address: dtromeur@cdcsp.univ-lyon1.fr (D. Tromeur-Dervout).
URL: http://cdcsp.univ-lyon1.fr/mathgrid (Y. Vassilevski).

[1] The work of this author was partially support by the GDR-CNRS MOMAS.

[2] This work is backwarded to the Région Rhône-Alpes thru the project: "Développement de méthodologies mathématiques pour le calcul scientifique sur grille".

such new systems. The development of such algorithms is difficult but the behaviour of the solution and the properties of operator can help to design latency-aware algorithms.

For example, the $C(p,q,j)$ time integration algorithms [10] performed at 80% efficiency on two distant computers linked with a 10 Mb/s communication bandwidth network. The target application solved Navier–Stokes equations coupled with reaction–diffusion equations. Aitken–Schwarz Domain Decomposition [11] is an another example of an algorithm design for metacomputer. It addresses the more significant challenge to build a fast solver for the Helmholtz operator and the reaction–convection–diffusion operator. This numerically efficient method delivers competitive performance on metacomputers with a standard Internet network. Successful experiments with more than 1280 processors split between 3 CrayT3E 450/375 MHz connected across the Atlantic with a 5 Mb/s bandwidth communication network are reported in [2].

From the engineering point of view, coupling huge high performance computers seems not to be realistic in terms of day-to-day practice and infrastructure costs. Nevertheless, due to material replacement staggering, industrial companies often have several medium computing resources with different performance characteristics. These computing resources can constitute a GRID architecture with high latencies and slow communication networks with fluctuating bandwidth shared by several users. The main drawback of this kind of computing architecture is that processor persistent availability is not guaranteed during computing.

The main objective of this paper is to propose and examine a method suitable for the GRID architecture. The method essentially accelerates the fully implicit solution of unsteady boundary value problems. The core of the methodology is the generation of a reduced model of lower dimension compared to the original problem. The implicit solution of the reduced model provides a much better initial guess for the inexact Newton backtracking (INB) algorithm. This results in a lesser number of nonlinear functions and preconditioner evaluations. The reduced model is generated on the basis of proper orthogonal decomposition (POD) [18,19] for the series of solutions already obtained.

Fully implicit approximations of the unsteady boundary value problems have become very popular in the computational community in the last decades. Their main appealing feature is the unconditional stability which is very important for long term simulations. The basic difficulty of fully implicit applications is the high arithmetical complexity of each time step caused by the solution of large systems of equations. In the case of nonlinear problems, the solution procedure is even more complicated. However, a set of robust (globally convergent) efficient nonlinear solvers was suggested in 90's [5,13,6,7]. Being a combination of the inexact Newton method and Krylov subspace iterations, the methods provide globally convergent algorithms (allowing large time steps) and efficient and simple implementations. In this paper we consider the application of the inexact Newton backtracking (INB) [6,7] algorithm for solving the unsteady driven cavity problem. For the implementation of the INB algorithm we refer to [16].

The outline of this paper is as follows. After a general description of a fully implicit discretisation in Section 2, we discuss shortly the inexact Newton backtracking algorithm in Section 3. Section 4 considers a convergence acceleration based on the choices of better initial guesses for Newton outer iterations. In Sections 4.1–4.3 we recall the POD, show how to generate the reduced model, and formulate our INB–POD algorithm. In Section 5 the implementation in GRID computations is discussed. Numerical experiments are reported in Section 6 where we formulate and advocate our unsteady model problems, present the performance of the standard algorithm INB and the proposed INB–POD, show the basic properties of its asynchronous parallelisation, and exhibit some gridding computation based on this algorithm.

2. Fully implicit discretisations of unsteady nonlinear problems

Let $L(u)$ be a nonlinear discrete operator representing a spatial approximation of a parabolic boundary value problem. The simplest robust technique for time approximation of unsteady problems is backward Euler time stepping:

$$\frac{u^i - u^{i-1}}{\Delta t} + L(u^i) = g^i. \tag{1}$$

The main advantage of the method is its unconditional stability. Being a first order scheme (in time), it may be generalised to higher order approximations (e.g., backward differences formulae). In general, the ith time step of a fully implicit scheme may be represented by the nonlinear system

$$F^i(u^i) = 0, \tag{2}$$

where F^i contains all the problem data and previous solutions. For instance, in the case of scheme (1),

$$F^i = u^i + L(u^i)\Delta t - u^{i-1} - g^i\Delta t.$$

The price to be paid for the robustness of the method is its arithmetical complexity: at each time step, a nonlinear system has to be solved. In the last decade, several robust nonlinear solvers have been proposed, analysed, and implemented [5,13,16]. However, the efficient solution for large nonlinear systems remains a challenge.

3. The inexact Newton backtracking (INB) method

Consider a nonlinear system

$$F(u) = 0.$$

The inexact Newton backtracking [6,7,16] method offers global convergence properties combined with potentially fast local convergence.

```
ALGORITHM INB.

LET $u_0$, $\eta_{\max} \in [0.5, 1)$, $t \in (0,1)$ AND $0 < \theta_{\min} < \theta_{\max} < 1$ BE GIVEN.
FOR $k = 0, 1, \dots$ (UNTIL CONVERGENCE) DO:
  SET INITIAL $\eta_k \in [0, \eta_{\max}]$ BY
    $\eta_k = \begin{cases} \min\left\{\eta_{\max}, \frac{|\|F(u_k)\| - \|F(u_{k-1}) + F'(u_{k-1})s_{k-1}\||}{\|F(u_{k-1})\|}\right\}, & k > 0, \\ 0.5, & k = 0. \end{cases}$   (3)
  FIND INITIAL $s_k$ SUCH THAT
    $\|F(u_k) + F'(u_k)s_k\| \leqslant \eta_k \|F(u_k)\|$,   (4)
  WHILE $\|F(u_k + s_k)\| > [1 - t(1 - \eta_k)]\|F(u_k)\|$ DO:
    CHOOSE $\theta \in [\theta_{\min}, \theta_{\max}]$.
    UPDATE $s_k \leftarrow \theta s_k$ AND $\eta_k \leftarrow 1 - \theta(1 - \eta_k)$.
  SET
    $u_{k+1} = u_k + s_k$.   (5)
```

Typical values of the parameters are as follows [16]: $\eta_{\max} = 0.9$, $\theta_{\min} = 0.1$, $\theta_{\max} = 0.5$, $t = 10^{-4}$. Eq. (3) gives one of the possible definitions of the "forcing term" which prescribes the accuracy of solving the system $F(u_k)s_k = -F'(u_k)$. The parameter t is used to judge sufficient reduction of $\|F(u_k)\|$. The judgement is based on the comparison of the actual function reduction $\|F(u_k) - F(u_k + s_k)\|$ and the reduction predicted by a local quadratic model $\|F(u_k)\| - \|F(u_k) + F'(u_k)s_k\|$:

$$\|F(u_k) - F(u_k + s_k)\| \geqslant t(\|F(u_k)\| - \|F(u_k) + F'(u_k)s_k\|).$$

Condition (4) holds for each s_k, η_k determined by the while-loop. Parameter $\theta \in [\theta_{\min}, \theta_{\max}]$ is chosen to minimalise a quadratic that interpolates $\|F\|$ in the direction of the inexact Newton step. The algorithm has a solid theoretical background [6,7]:

Theorem. *Let F be continuously differentiable. If $\{u_k\}$ produced by the Algorithm INB has a limit point u such that $F'(u)$ is invertible, then $F(u) = 0$ and $u_k \to u$. Furthermore, if F' is Lipschitz continuous at u, then*

$$\|u_{k+1} - u\| \leqslant \beta \|u_k - u\| \|u_{k-1} - u\|, \quad k = 1, 2, \dots$$

for a constant β independent of k.

Inequality (4) implies the approximate iterative solution of the Newton step system

$$F'(u_k)s_k = -F(u_k) \tag{6}$$

with relative reduction of the residual (for a trivial initial guess) η_k. The forcing term η_k (3) is chosen dynamically to avoid over-solving the systems (6). Backtracking is used to globalise the convergence, and updated s_k and η_k always satisfy (4). The iterative solution of (6) requires only evaluation of $F'(u_k)$ on a vector. This allows us to replace the $F'(u_k)v$ by its finite difference approximation, *e.g.*,

$$F'(u_k)v = \frac{1}{\delta}[F(u_k + \delta v) - F(u_k)]. \tag{7}$$

Hereinafter, the GMRES(30) method is used to obtain the solution iteratively.

The algorithm INB presumes the choice of an initial guess u_k for nonlinear iterations. We recall that the arithmetical complexity of the method is expressed in the total number of function evaluations n_{evF} and the total number of preconditioner evaluations n_{evP} (if any); the remaining overheads are negligible.

4. INB acceleration via model reduction

4.1. Proper orthogonal decomposition

Proper orthogonal decomposition (POD) provides a way to find optimal lower dimensional approximations of a given series of data. More precisely, it produces an orthonormal basis for representing the data series in a certain least squares optimal sense [18,19]. Combined with the Galerkin projection, POD is a tool for the generation of reduced models of lower dimension. The reduced models may give a better initial guess for the Newton solution at the next time step.

POD provides a definite answer to the question: Which of the m-dimensional subspaces $S \subset \mathbf{R}^N$ is the most close (in the terms of the least squares) to the given set of vectors $\{u^i\}_{i=1}^n$,

$$S = \arg \min_{S \in \mathbf{R}^{N \times m}} \sum_{i=1}^{n} \|u^i - P_S u^i\|^2?$$

Here P_S is the orthogonal projection onto S. Define the correlation matrix $R = XX^{\mathrm{T}}$, $X = \{u^1 \cdots u^n\}$, and find m eigenvectors of the problem

$$Rw_j = \lambda_j w_j, \quad \lambda_1 \geqslant \cdots \geqslant \lambda_N \geqslant 0$$

corresponding to m largest eigenvalues $\lambda_1 \geqslant \cdots \geqslant \lambda_m$. Then

$$S = \operatorname{span}\{w_j\}_{j=1}^m \tag{8}$$

and

$$\sum_{i=1}^{n} \|u^i - P_S u^i\|^2 = \sum_{j=m+1}^{N} \lambda_j. \tag{9}$$

The computational cost of finding m-largest eigenvalues of symmetric matrix R is not high. Indeed, our experience shows that for $m = \mathrm{O}(10)$ the application of the Arnoldi process requires a few tens of R-matrix–vector multiplications in order to retrieve the desirable vectors with very high accuracy [14]. In spite of the large dimensions of N and the density of R, the matrix–vector multiplication is easy to evaluate, due to the factored representation $R = XX^{\mathrm{T}}$. Let $X \in \mathbf{R}^{N \times n}$, $a \in \mathbf{R}^n$ and $b \in \mathbf{R}^N$ then the arithmetical cost of the evaluation Xa (and $X^{\mathrm{T}}b$) is Nn multiplications and not more than Nn additions, therefore, R-matrix–vector multiplication costs at most $4Nn$ flops.

Since the method of finding $\{w_j\}_{j=1}^m$ is based solely on matrix–vector multiplication, it may be easily parallelised

[15]. Taking into account the factored form of the matrix, we parallelise the method as follows. Let rows of X be split into p groups according to the value mod$(i-1,k)$, where i is the row index and $k = N/p$. Each group is allocated on a processor with the same index. Let a vector $b \in \mathbf{R}^N$ be allocated among the processors according to the same rule. Then $\mathbf{R}^n \ni a = X^{\mathrm{T}} b$ is obtained by the global summation of all local contributions (MPI_ALLREDUCE), whereas Xa is evaluated by each processor independently of the others.

Other details of numerical implementation and parallelisations will be given below.

4.2. Generation and solution of the reduced model

Each time step of the scheme (1) generates Eq. (2) which we call the original model. A reduced model is generated on the basis of POD for a sequence of solutions at time steps $\{u^i\}_{i=i_b}^{i_e}$, $i_e - i_b + 1 = n$. The eigenvectors $\{w_j\}_{j=1}^{m}$ may be considered as the basis of m-dimensional subspace whose projection is $V_m = \{w_1 \cdots w_m\} \in \mathbf{R}^{N\times m}$. The reduced model is the Galerkin projection onto this subspace:

$$V_m^{\mathrm{T}} F^i(V_m \hat{u}^i) = 0, \tag{10}$$

or, equivalently,

$$\widehat{F}^i(\hat{u}^i) = 0, \tag{11}$$

where the unknown vector $\hat{u}^i \in \mathbf{R}^m$ and $\widehat{F}^i : \mathbf{R}^m \to \mathbf{R}^m$.

The reduced model is the very low dimension m nonlinear equation. For its solution, we adopt the same INB algorithm with the finite difference approximation of the Jacobian–vector multiplication. Being the formal Galerkin projection, each evaluation of function $\widehat{F}^i(\hat{u}_k^i)$ is the sequence of the following operations: $u_k^i = V_m \hat{u}_k^i$, $f_k^i = F^i(u_k^i)$, $\hat{f}_k^i = V_m^{\mathrm{T}} f_k^i$. Therefore, the overhead is matrix–vector multiplications for V_m and V_m^{T}, *i.e.*, 4 N m flops. We notice that usually $m = \mathrm{O}(10)$ and the evaluation of function $F(u)$ is much more expensive than 40 N m which implies a negligible weight of overheads.

Another important consequence of low dimensionality of (11) is that the INB algorithm may be applied without any preconditioner. Indeed, were the function $F()$ linear in the vicinity of u_k (rf. (6)), the GMRES iterations would converge within at most m iterations (provided m is larger than the Krylov subspace dimension). Since $\delta \ll 1$ in (7), $F()$ is very close to linear operator and m GMRES iterations must provide at most $\mathrm{O}(\delta)$ accuracy. For $\delta \ll \epsilon$, the stopping tolerance for the INB algorithm, convergence is implied within m iterations for each system (6).

4.3. A fully implicit solver with POD-reduced model acceleration

Coupling POD and Galerkin projection for the generation of the reduced model gives a powerful tool for acceleration of the fully implicit schemes. Let n, the length of data series, be defined, as well as the desirable accuracy ϵ for F^i: $\|F^i(u^i)\| \leqslant \epsilon$. For any time step $i = 1, \ldots,$ perform:

```
ALGORITHM INB–POD

IF $i \leqslant n$, THEN
  SOLVE $F^i(u^i) = 0$ BY PRECONDITIONED INB
  WITH THE INITIAL GUESS $u_0^i = u^{i-1}$ AND ACCURACY $\epsilon$
ELSE
1. IF(mod$(i,n) = 1$): !(SEARCH FOR THE REDUCED BASIS)
     (A) FORM $X = \{u^{i-n} \cdots u^{i-1}\}$;
     (B) FIND SO MANY LARGEST EIGENVECTORS $w_j$ OF
         $R = XX^{\mathrm{T}}$ THAT
             $\sum_{j=m+1}^{N} \lambda_j \leqslant \epsilon$;
     (C) FORM $V_m = \{w_1 \cdots w_m\}$
2. !(COMPUTE THE BETTER INITIAL GUESS BY POD)
     (A) SET $\hat{u}_0^i = V_m^{\mathrm{T}} u^{i-1}$
     (B) SOLVE $\widehat{F}^i(\hat{u}^i) = 0$ BY NONPRECONDITIONED INB
         WITH THE INITIAL GUESS $\hat{u}_0^i$ AND ACCURACY $\epsilon/10$
3. !(SOLVE THE FULL PROBLEM WITH THE BETTER INITIAL
   GUESS)
     (A) SET $u_0^i = V_m \hat{u}^i$
     (B) SOLVE $F^i(u^i) = 0$ BY PRECONDITIONED INB WITH
         THE INITIAL GUESS $u_0^i$ AND ACCURACY $\epsilon$
```

Several remarks can be made as follows. The absence of the preconditioner for the reduced model is dictated by two things: (a) it is not clear how to construct a preconditioner for the reduced model, (b) it is not required if m is small. The reduced model is slightly over-solved: this provides the better initial guess u_0^i. The number of eigenvectors is chosen adaptively in the above algorithm: it allows us to form a reduced model that approximates the original model with the desirable accuracy ϵ. Actually, this condition may be replaced by a rougher $N\lambda_{m+1} < \epsilon$ or even a fixed number m, $m = 10$–40. The solution of the eigenvalue problem may be performed asynchronously with the implicit solution: as soon as V_m is formed, the reduced model becomes the active substep. The latter observation creates a lot of possibilities for organising the computations. See the next section.

5. The POD-reduced model acceleration and its GRID applications

The present acceleration is well designed for GRID computing. The target architecture is represented by a large amount of low cost computational resources (clusters) connected via a standard Ethernet communication network. We consider the basic features of the GRID architecture and appropriate modes of POD usage in this context:

- A slow communication network with high latency time between clusters of resources. It is usual to have a one or two orders of magnitude gap between the communication speeds inside and outside a cluster. The POD acceleration can be used wherever POD data are available. The asynchronous nonblocking communications between the POD generator and the solver resource provide computation of the time step without idling. Therefore, a slow network with high latency time is affordable for the proposed technology.

- A high probability of failure of an element of the computing resources. This is typical for GRIDs where the resources are not dedicated to a single application and a single user. The conventional way to cope with a hardware failure is on-fly backup of the solution which deteriorates the performance. Being activated on a separate computational resource, the POD generator provides a natural way to restart the computation on the solver resource. The MPI-2 process model allows us to spawn a new set of MPI processes. The POD can be the external resource control that can start the spawn processes of the application, in case the solver resources fail. Upon an interruption of the data supply, the POD generator can restart the solver processes and deliver them the appropriate last solution recovered from the reduced model representation.
- The POD generator task is waiting for data from the solver resource and will compute the POD basis when sufficient data are gathered. For the sake of more efficient use of the POD generation resource, it may work with other tasks as well. For instance, the POD generator can be used by several solvers and perform the POD on different sets of solutions tagged by the generating solver. In addition, the reduced basis may be used for other postprocessing tasks such as data visualisation or a posteriori error estimation.

6. Numerical experiments

6.1. Formulation of model problems

In order to illustrate the basic features of the proposed methodology, we choose the backward Euler approximation of the unsteady 2D Navier–Stokes equations. We consider the classical driven cavity problem in the streamfunction-vorticity formulation [8]:

$$\frac{\partial\omega}{\partial t}-\frac{1}{Re}\Delta\omega+(\psi_y\omega_x-\psi_x\omega_y)=0 \quad \text{in } \Omega, \tag{12}$$

$$\begin{aligned}&-\Delta\psi=\omega \quad \text{in } \Omega,\\ &\psi|_{t=0}=0 \quad \text{in } \Omega,\\ &\psi=0 \quad \text{on } \partial\Omega,\\ &\left.\frac{\partial\psi}{\partial n}\right|_{\partial\Omega}=\begin{cases} v(t) & \text{if } y=1,\\ 0 & \text{if } 0\leqslant y<1.\end{cases}\end{aligned} \tag{13}$$

Here, $\Omega=(0,1)^2$, $Re=1000$, and $v(t)$ is the unsteady boundary condition leading the flow. After elimination of vorticity, we obtain the streamfunction formulation:

$$\begin{aligned}&\frac{\partial}{\partial t}(\Delta\psi)-\frac{1}{Re}\Delta^2\psi+(\psi_y(\Delta\psi)_x-\psi_x(\Delta\psi)_y)=0 \quad \text{in } \Omega,\\ &\psi|_{t=0}=0 \quad \text{in } \Omega,\\ &\psi=0 \quad \text{on } \partial\Omega,\\ &\left.\frac{\partial\psi}{\partial n}\right|_{\partial\Omega}=\begin{cases} v(t) & \text{if } y=1,\\ 0 & \text{if } 0\leqslant y<1.\end{cases}\end{aligned} \tag{14}$$

Three different types of flows are simulated: saturating (to the steady solution), quasi-periodic in time, and quasi-periodic in time with variable periods (arrhythmic). These three cases are defined by the unsteady boundary velocity $v(t)$. We set $v(t)=1+(t+10)^{-1}$ for the saturating flow, $v(t)=1+0.2\sin(t/10)$ for the quasi-periodic flow, and $v(t)=1+0.2\sin([1+0.2*\sin(t/5)]*t/10)$ (see Fig. 1). We motivate the chosen parameters as follows. In the case of $v(t)=1$, the unsteady solution saturates within $t_s\sim150$. Therefore, to get a quasi-periodic solution, we need the periodic forcing term with the period $T<t_s$ but comparable with t_s, $T\sim t_s$. Indeed, if $T\ll t_s$, the inertia of the dynamic system will smear out the amplitude of the oscillations; if $T>t_s$, the dynamic system will have enough time to adapt to the periodic forcing term and demonstrate periodic behavior. The function $\sin(t/10)$ has the period $T=20\pi$ which perfectly fits the above restrictions for the quasi-periodicity. The arrhythmic flow is a simple modification of the quasi-periodic flow by arrhythmic scaling of the time $t\to[1+0.2*\sin(t/5)]*t$. It is well known that the feasible time step Δt for approximation of periodic solutions satisfies $12\Delta t=T$, *i.e.*, $\Delta t\sim5$. Therefore, in the case of $v(t)=1$, the saturation will occur within 30 time steps. This is not enough for the demonstration of the POD-acceleration. Therefore, we artificially extend the saturation time by choosing the unsteady saturating boundary condition $v(t)=1+(t+10)^{-1}$.

For the discretisation in time, we chose the backward Euler scheme (1). For the discretisation in space (developed by P. Brown), we adopt the P_1 finite element spaces applicable to the biharmonic problems as it was shown in [12].

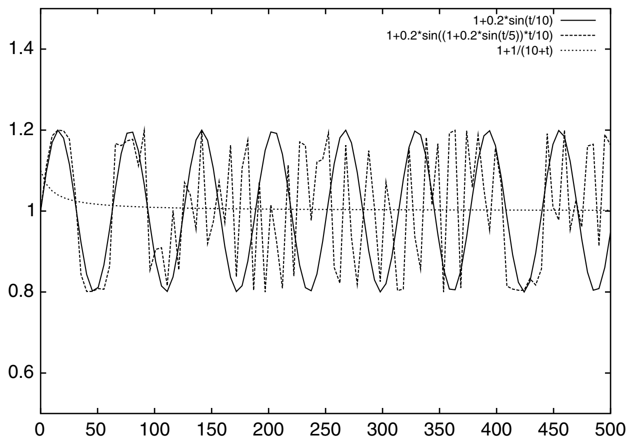

Fig. 1. Different types of boundary velocity $v(t)$.

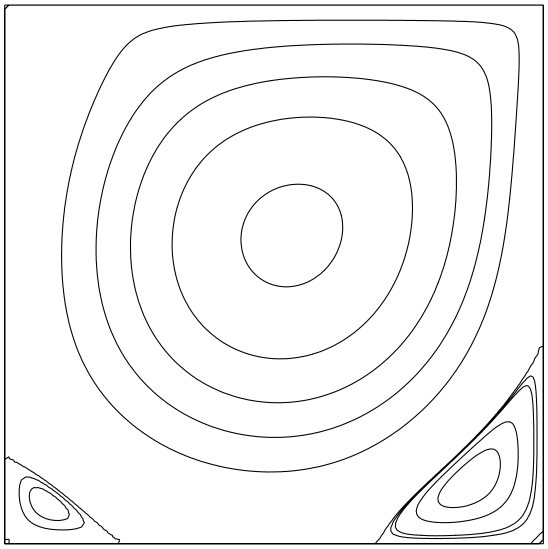
Fig. 2. Streamfunction isolines $\psi = -0.1, -0.08, -0.06, -0.04, -0.02, 0, 0.00005, 0.0001, 0.0005, 0.001, 0.0025$ for quasi-periodic solution at time $t = 500$, $h = 2^{-7}$.

This discretisation is also equivalent to that obtained using standard finite differences. We consider two uniform grids with mesh steps $h = 2^{-7}, 2^{-8}$ providing 16,129 and 65,025 unknowns, respectively. Typical quasi-periodic solution isolines are shown in Fig. 2. The nonlinear problems (2) are solved by the NITSOL package [16] with the preconditioned GMRES(30) iterations. The stopping criterion for the INB algorithm is $\|F^i(u^i_k)\| < 10^{-7}\|F^0(0)\|$. For the preconditioner, we use the discretised version of $P = \frac{1}{Re}\Delta^2$. For a uniform mesh, systems of the form $Pw = u$ can be solved very efficiently using a fast solver [4]. For small and moderate Reynolds numbers, this preconditioner provides independence of the convergence rate of the mesh size. Evaluations of the Jacobian are replaced by the finite difference approximations of the first order. Therefore, the dominant contribution to the arithmetical work is given by the number of function evaluations n_{evF} and number of preconditioner evaluations n_{evP} for each time step.

The sequential and parallel numerical experiments presented in Sections 6.2–6.4 have been performed on alpha ev67 processors with 8Mo L2-Cache, cadenced to 667 MHz, with 800 Mb/s communication bandwidth network.

6.2. Computational performance of the standard implicit solver

In Table 1 we present the performance of the standard NITSOL solver with the initial guess equal to the solution at the previous time step. The number of function and preconditioner evaluations does not depend on the mesh size. Consequently, the CPU time (for a time step) increases by factor 5 when the number of unknowns is multiplied by factor 4. Slight nonproportionality is attributable to the arithmetical complexity of the preconditioner evaluation.

Table 1
Performance of the algorithm INB (NITSOL) for the three types of unsteady solution at several time steps (i) and on two meshes (h), $u^i_0 = u^{i-1}$

	$h = 2^{-7}$			$h = 2^{-8}$		
	$i = 10$	$i = 20$	$i = 30$	$i = 10$	$i = 20$	$i = 30$
Saturating solution						
$\|F^i(u^i_0)\|$	0.005	0.0014	0.0006	0.015	0.004	0.0018
n_{evF}	127	107	96	118	93	83
n_{evP}	122	102	91	113	88	78
CPU time	2	1.73	1.54	9.7	7.3	7
Quasi-periodic solution						
$\|F^i(u^i_0)\|$	0.13	0.28	0.03	0.36	0.79	0.09
n_{evF}	150	191	172	166	186	189
n_{evP}	144	185	167	160	180	183
CPU time	2.4	3.1	2.9	13.4	15.3	16.1
Arrhythmic solution						
$\|F^i(u^i_0)\|$	0.005	0.77	0.46	0.01	2.2	1.3
n_{evF}	122	215	201	115	205	227
n_{evP}	117	208	195	110	198	221
CPU time	2.0	3.5	3.3	9.2	16.7	19.5

However, both n_{evF} and n_{evP} may depend on the time step: for saturating solution, they decrease monotonically, whereas for the other cases they may oscillate within certain ranges. The reason is evident: in the first case, the choice $u^i_0 = u^{i-1}$ provides better initial guess ($\|F^i(u^i_0)\|$) as $t \to \infty$, while in the other cases the quality of the initial guess $u^i_0 = u^{i-1}$ depends on the time moment t (see Fig. 1).

6.3. Sequential runs: speed-up of computation by POD and its basic features

We consider the performance of the algorithm INB–POD with the following parameters: the data (solutions) series are $\{u^{20k-10} \cdots u^{20k+9}\}$, $k = 1, 2, \ldots$, so $n = 20$, and the dimension of the reduced model is fixed to $m = 10$. In Tables 2 and 3 we present the arithmetical complexity of certain time steps, in terms of n_{evF}, n_{evP} and the CPU time, as well as the quality of the initial guess $\|F^i(u^i_0)\|$ due to the reduced model. The first observation is that the acceleration is significant, ~2–5-fold in comparison with the standard algorithm INB. This is due to a much better initial guess for the original model solver (cf. $\|F^i(u^i_0)\|$). Due to the super-linear convergence of the INB algorithm, this results in smaller values of n_{evF}, n_{evP}. The price to be paid for this reduction is the cost of the reduced problem solution. As mentioned before, the complexity of the function $\widehat{F}$ evaluation only slightly exceeds that for F, whereas the number of preconditioner evaluations is zero for the reduced model. Since in the considered application (as in the majority of applications), the complexity of the preconditioner evaluation dominates the complexity of the function evaluation, the speed-up is attributable to the ratio of n_{evP} for the standard algorithm and the accelerated one. As can be seen from the tables, this ratio depends on the type of unsteady solution and on the time moment. For the saturating and quasi-periodic solutions, the reduced model is capable of recovering the solution at

Table 2
Performance of the algorithm INB–POD (NITSOL) for the saturating solution at several time steps (i) and on two meshes (h), $u_0^i = V_m\hat{u}^i$

	$h = 2^{-7}$			$h = 2^{-8}$		
	$i = 32$	$i = 42$	$i = 52$	$i = 32$	$i = 42$	$i = 52$
$\|F^i(u_0^i)\| \times 10^{-7}$	4.4	15	7	14	37	12
n_{evF}	25 + 12	25 + 30	24 + 19	27 + 14	27 + 27	25 + 12
n_{evP}	0 + 10	0 + 27	0 + 16	0 + 12	0 + 24	0 + 10
CPU time	0.1 + 0.16	0.1 + 0.4	0.1 + 0.3	0.7 + 0.9	0.7 + 1.8	0.6 + 0.7

The first entry of each sum corresponds to the contribution of the reduced model, the second is due to the original model.

Table 3
Performance of the algorithm INB–POD (NITSOL) for the nonsaturating solutions at several time steps (i) and on two meshes (h), $u_0^i = V_m\hat{u}^i$

	$h = 2^{-7}$			$h = 2^{-8}$		
	$i = 32$	$i = 52$	$i = 72$	$i = 32$	$i = 52$	$i = 72$
Quasi-periodic solution						
$\|F^i(u_0^i)\| \times 10^{-6}$	25	0.9	0.5	22	1	2.6
n_{evF}	54 + 74	45 + 22	44 + 11	44 + 55	45 + 11	44 + 19
n_{evP}	0 + 69	0 + 19	0 + 9	0 + 51	0 + 9	0 + 16
CPU time	0.2 + 1.1	0.2 + 0.3	0.2 + 0.15	1.2 + 4.2	1.1 + 1.1	1.1 + 1.2
Arrhythmic solution						
$\|F^i(u_0^i)\| \times 10^{-4}$	4	3.6	1.7	5.5	4.3	1.8
n_{evF}	50 + 110	51 + 96	49 + 94	50 + 98	42 + 80	49 + 77
n_{evP}	0 + 105	0 + 91	0 + 89	0 + 93	0 + 75	0 + 72
CPU time	0.2 + 1.7	0.2 + 1.5	0.2 + 1.5	1.3 + 7.7	1.0 + 6.4	1.1 + 5.9

The first entry of each sum corresponds to the contribution of the reduced model, the second is due to the original model.

the next time step very accurately ($\|F^i(u_0^i)\| \sim 10^{-6}$) which provides the essential reduction of n_{evP} (and n_{evF}). However, at time step 32 the quasi-periodic flow is not yet very well stabilised, and the prediction of the reduced model is not as good ($\|F^i(u_0^i)\| \sim 10^{-5}$) yielding only 2-fold acceleration. For the arrhythmic flow, the quality of the prediction is even worse ($\|F^i(u_0^i)\| \sim 10^{-4}$), and the typical acceleration factor is 2–3. At the early stages of the arrhythmic flow the acceleration may be small, as observed for time step 32. Here, the solution from the previous time step occasionally turns out to be almost as good as the initial guess predicted by the reduced model, ~0.01 versus ~0.001. As a result, the reduction of n_{evP} is only 10–20%, and the overall acceleration is insignificant. On the other hand, the complexity of the reduced model solution does not vary considerably: n_{evF} ranges from 25 to 50 for all the cases. The reason is the low dimensionality ($m = 10$) of the reduced problem.

We remark here that the arrhythmic flow is a very stiff test for the methodology: the solution at the next time step can hardly be approximated by a composition of several solutions at previous time steps. Consequently, the POD-reduced model is not capable of providing a very good initial guess and many-fold speed-ups. Could the extension of the data (solution) series, n, and/or enrichment of the reduced model basis, m, increase the speed-up? Controversial tendencies do not allow us to answer the question *a priori*: the larger m is, the better the prediction of the reduced model, but its quality is limited by the accuracy of the representation of u^{i+1} via the series $\{u^{i_b} \cdots u^{i_e}\}$; on the other hand, the complexity of the reduced model solution increases for large m. Therefore, the answer depends on the particular problem and parameters and must be based on numerical evidence. For the case of the arrhythmic solution, we present, in Table 4, the complexity of certain time steps for several pairs m,n. The 2-fold increase of m does provide a better initial guess but the increased cost of the reduced model solution is not compensated by the speed-up of the original model solution. On the other hand, the 2-fold increase of n does not provide a better resolution of the reduced model: due to the irregularity (in time) of the unsteady solution, the extension of the data series may not provide a better approximation. Moreover, the extension of data series may slightly deteriorate the performance of the algorithm INB–POD. The reason is that the more “useless” solutions contribute to the basis of the reduced model, the less adequate the reduced model (with fixed m). We notice that for unsteady problems with “predictable” solutions (saturating or quasi-periodic) the positive effects of the increase of n may be more pronounced, with the exception that for the quasi-periodic solution the choice $n > T/\Delta t$ does not seem to be feasible. The increase of m is always constrained by the compromise between the cost of the solution of the reduced model and the actual speed-up of the original model.

6.4. Asynchronous runs on a parallel computer

An important feature of the algorithm INB–POD is that it is not strictly sequential: computation of the basis for the reduced model may be performed on another processor. The implicit solution may be advanced without the reduced

Table 4
Performance of the algorithm INB–POD (NITSOL) for the arrhythmic solution at several time steps (i), $u_0^i = V_m\hat{u}^i$

	$i=32$	$i=52$	$i=72$	$i=112$	$i=152$
$m=10$, $n=20$, $X=\{u^{20k-10}\cdots u^{20k+9}\}$					
$\|F^i(u_0^i)\|\times 10^{-4}$	5.5	4.3	1.8	6	1.4
n_{evF}	50 + 98	42 + 80	49 + 77	48 + 101	51 + 78
n_{evP}	0 + 93	0 + 75	0 + 72	0 + 96	0 + 73
CPU time	1.3 + 7.7	1.0 + 6.4	1.1 + 5.9	1.1 + 8.0	1.2 + 6.0
$m=20$, $n=20$, $X=\{u^{20k-10}\cdots u^{20k+9}\}$					
$\|F^i(u_0^i)\|\times 10^{-4}$	0.5	0.4	2.0	0.4	0.4
n_{evF}	101 + 67	96 + 61	99 + 80	107 + 69	86 + 58
n_{evP}	0 + 63	0 + 57	0 + 75	0 + 65	0 + 54
CPU time	4.0 + 5.2	4.2 + 4.8	4.2 + 6.3	4.7 + 5.3	3.5 + 4.5
$m=10$, $n=40$, $X=\{u^{40k-30}\cdots u^{40k+9}\}$					
$\|F^i(u_0^i)\|\times 10^{-4}$		3.9	5.5	6.0	3.6
n_{evF}		42 + 86	52 + 93	58 + 97	45 + 90
n_{evP}		0 + 81	0 + 88	0 + 92	0 + 85
CPU time		1.0 + 6.7	1.2 + 7.2	1.3 + 7.6	1.1 + 7.1
$m=20$, $n=40$, $X=\{u^{40k-30}\cdots u^{40k+9}\}$					
$\|F^i(u_0^i)\|\times 10^{-4}$		0.5	0.7	0.5	0.3
n_{evF}		86 + 59	100 + 64	109 + 70	87 + 54
n_{evP}		0 + 55	0 + 60	0 + 66	0 + 50
CPU time		4.0 + 4.5	4.5 + 4.9	4.8 + 5.4	3.6 + 4.1

The first entry of each sum corresponds to the contribution of the reduced model, the second is due to the original model, $h = 2^{-8}$.

model POD acceleration, or with the obsolete (not updated) reduced model acceleration. The technology of the asynchronous data exchanges provides the tools for launching the reduced model usage only when the respective reduced basis is available (*i.e.*, computed somewhere else and received by the processor carrying out the implicit solution). On the other hand, upon each time step termination the solution may be sent asynchronously to the processor resolving eigenproblems. Therefore, the number of time steps n_{delay} accelerated with obsolete data (or not accelerated) depends on the time for solving the partial eigenproblem for matrix $R = XX^{\text{T}}$ and the time of data (solution series and reduced model basis) exchange. For parallel computers the time for data exchange is small ($N + mN$) in comparison with the the complexity of the partial eigenproblem solution. Hence, it is the eigensolver that affects the value of n_{delay}. In Table 5 we present n_{delay} for all types of unsteady solutions on the mesh $h = 256^{-1}$ when the partial eigenproblem is solved on one processor while the implicit time stepping is realised on another processor. It is seen that asynchronous exchanges affect one or two time steps.

In order to minimise n_{delay}, one can reduce the time of the partial eigensolution by its parallelisation, since it is based on the R-matrix–vector multiplication [14]. Due to the factored form of $R = XX^{\text{T}}$, the R-matrix–vector multiplication is very easy to parallelise if each solution vector is distributed over the set of available processors. The full parallel code using BLAS routines is trivial:

```
1. subroutine av(X, ldx, n, m, v, w, buf, comm)
2. include 'mpif.h'
3. integer n, m, ldx, comm
4. Double precision X(ldx,m), v(n), w(n), buf(*)
5. Double precision one, zero
6. parameter (one = 1D0, zero = 0D0)
7. Integer ierr
8. call dgemv('T', n, m, one, X, ldx, v, 1, zero, w, 1)
9. call MPI_ALLREDUCE(w, buf, m,MPI_DOUBLE_PRECISION, & MPI_SUM,comm, ierr)
10. call dgemv('N', n, m, one, X, ldx, buf, 1, zero, w, 1)
11. return
12. end
```

Table 5
Number of time steps using the obsolete data or no data because of asynchronous data exchanges between the two processors (advancing the implicit scheme and solving the partial eigenproblem), $n = 20$, $m = 10$, $h = 256^{-1}$

	Update reduced model at time step i			
	32	52	72	92
Saturating sol.				
n_{delay}	1	2	2	2
Quasi-periodic sol.				
n_{delay}	1	2	2	2
Arrhythmic sol.				
n_{delay}	1	1	1	1

Remarkably, this simple code provides a parallel efficiency larger than 1, as can be seen from Table 6. The only reason for this may be cash effects.

6.5. Comparison in the GRID framework

GRID experiments involve a computer A (SGI Altix350 with Ithanium 2 processors cadenced at 1.3 GHz/3 Mo,

Table 6
Speed-up of the parallelisation of the eigensolver PARPACK [15]

#Procs for eig. pr.	1	2	3
Time of the eig. sol.	8.5	3.4	1.4

Quasi-periodic solution, $n = 20$, $m = 10$, $h = 256^{-1}$, 30th time step.

1.3 Gb/s network bandwidth) and a computer B (6 nodes Linux cluster with AMD BiAthlon 1600+ MP processors, with 256KB L2-cache, 1GB of RAM per node, 100 Mb/s Ethernet internal network). The MPICH communication library is compiled with the ch_p4 interface. A latency of 140 μs and a maximum bandwidth of 71 Mb/s have been measured for the communication network between computers A and B. For compilation we use the g77 Fortran compiler.

This section compares the conventional solver INB with the initial guess from the previous time step, the INB–POD solver running on computer A, and the INB solver running on computer A and communicating with the POD generator running on computer B. The POD acceleration begins at the 30th time step.

Figs. 3 and 4 give the elapsed time of the INB solver for the arrhythmic and the periodic cases on the GRID architecture context. The figures show that:

- the INB–POD gives quite similar results when it performs on the GRID context or on homogeneous computer. Consequently the computer B can be used with no penalty on the performances;
- the elapsed time is greater than that in Tables 1–4 due to the poor performance of the g77 compiler on IA64 processor. Nevertheless, the acceleration factor of the INB–POD remains high.

Figs. 5 and 6 represent the GRID computation in detail. In addition to the total elapsed time of each time step, they

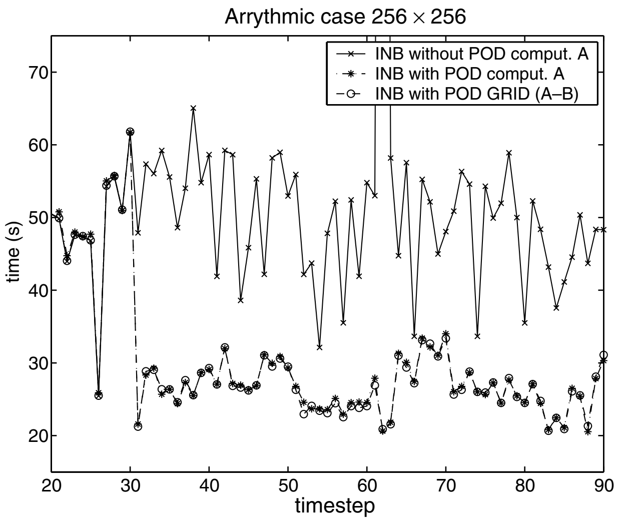

Fig. 3. Arrhythmic case: comparison of elapsed time of INB on computer A, INB with POD initial guess computer A, INB with POD initial guess GRID computer A (POD computed on computer B).

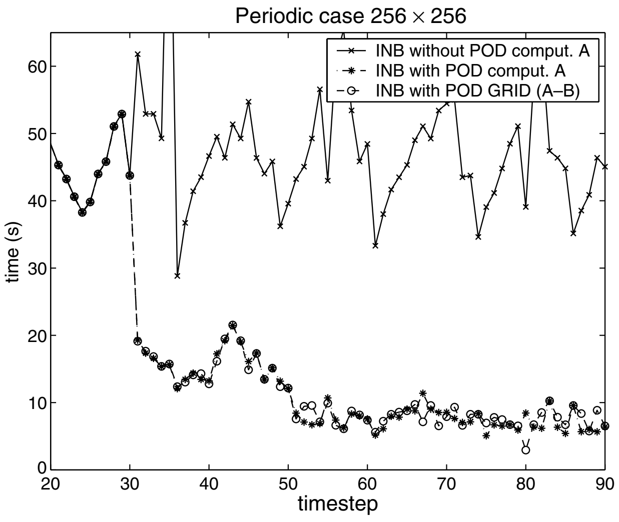

Fig. 4. Periodic case: comparison of elapsed time of INB on computer A, INB with POD initial guess computer A, INB with POD initial guess GRID computer A (POD computed on computer B).

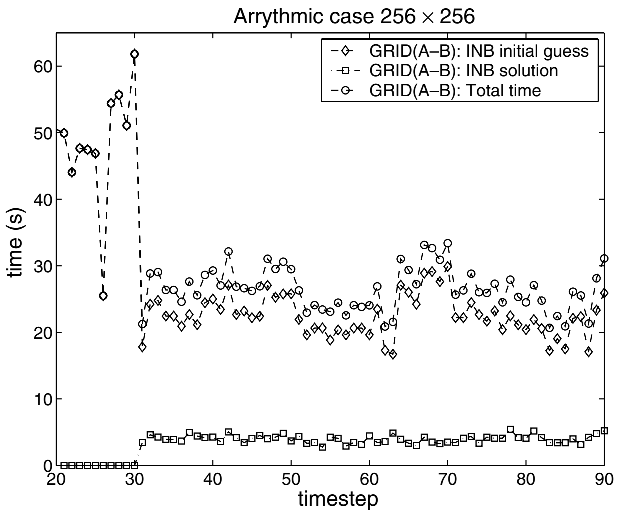

Fig. 5. Elapsed times for the GRID computation of the arrhythmic case: total time for the time step, time of computation of the initial guess by INB, time of the INB solution of the original model.

exhibit the time t_1 spent for the initial guess computation by INB applied to the reduced model, and the time t_2 of the INB solution of the original model. We observe that t_1 is almost constant with the mean value 3.7 s for the periodic case and 3.9 s for the arrhythmic case. In contrast, t_2 manifests strong (more than 100%) fluctuations with the mean value 6.6 s for the periodic case and 22.5 s for the arrhythmic case. The stability of t_1 is the consequence of the stable convergence of INB due to the low dimension ($m = 10$) of the reduced model.

We experiment with the counterpart of Table 5 for the GRID computations. We consider two configurations of resources. Configuration I uses one processor for NITSOL and two processors for POD on computer A.

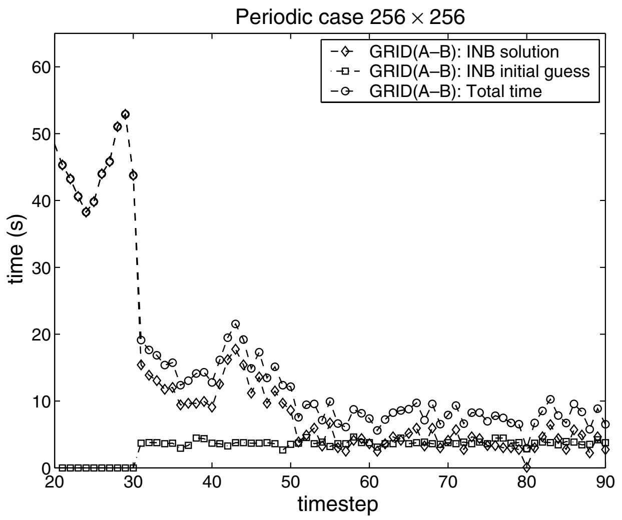

Fig. 6. Elapsed times for the GRID computation of the periodic case: total time for the time step, time of computation of the initial guess by INB, time of the INB solution of the original model.

Configuration II uses one processor for NITSOL on computer A and four processors for POD on computer B. The mesh size is $h = 128^{-1}$. In both cases the number of delayed time steps n_{delay} is equal to 1. The main reason for such a good efficiency is greater computational time compared to the alpha processor: the POD generator has more time to compute and send the reduced basis to the INB solver even on a coarser mesh. These results demonstrate the appealing features of the approach in day-to-day engineering computations. First, it is possible to run the POD and the INB on the same processor. However, there exists the risk of performance deterioration due to cache reuse and/or code swapping as the POD needs sufficient data to be relevant. Second, the cost of computer B processor dedicated to the POD is much less that of computer A processor. Consequently, it is beneficial to load the computer B with the useful but not critical task of POD generation and dedicate the advanced processors to heavy computation of nonlinear systems. Third, the spare processors on computer B can postprocess the solution (a posteriori error estimates, visualisation). Fourth, the spare processors can monitor the computation and restart it in case of failure of computer A.

We note that the reported GRID implementation suggests certain improvements. MPICH-Madeleine [1] should allow us to use different MPICH communications protocols both inside and outside a cluster. Besides, the employment of the Intel Fortran compiler on the IA64 processor architecture should improve the performance of the INB–POD solver.

7. Conclusions

The method for the acceleration of the fully implicit solution of nonlinear boundary value problems is presented. The use of the reduced model to a compute much better initial guess reduces the computational time as well as the numbers of nonlinear and linear iterations.

The asynchronous communications with the POD generator make the algorithm appealing for GRID computing [3]. The parallel efficiency in the GRID context is understood as the computational speed-up on the solver resource due to communications with the POD generator, and not the parallel speed-up with respect to the number of processors.

Another appealing feature of the approach is its hardware failure protection. Upon failure, the solver resource can recover its computation by spawning the MPI processes and using the POD data from the POD generator resource.

References

[1] Aumage O, Bougé L, Eyraud L, Mercier G, Namyst R, Prylli L, et al. High performance computing on heterogeneous clusters with the madeleine II communication library. Cluster Comput 2002;5(1):43–54.
[2] Barberou N, Garbey M, Hess M, Resch M, Rossi T, Toivanen J, et al. Efficient meta-computing of elliptic linear and non-linear problems. J Parallel Distributing Comput 2003;63:564–77.
[3] Beaugendre P, Priol T, Alléon G, Delavaux D. A client/server approach for HPC applications within a networking environment. In: Proceedings of HPCN'98. LNCS, vol. 1401. Amsterdam, The Netherlands: Springer-Verlag; 1998. p. 518–25.
[4] Bjorstad P. Fast numerical solution of the biharmonic Dirichlet problem on rectangles. SIAM J Numer Anal 1983;20:59–71.
[5] Brown P, Saad Y. Hybrid Krylov methods for nonlinear systems of equations. SIAM J Sci Stat Comput 1990;11:450–81.
[6] Eisenstat S, Walker H. Globally convergent inexact Newton methods. SIAM J Optim 1994;4:393–422.
[7] Eisenstat S, Walker H. Choosing the forcing terms in an inexact Newton method. SIAM J Sci Comput 1996;17:16–32.
[8] Feistauer M. Mathematical methods in fluid dynamics. Pitman monographs and surveys in pure and applied mathematics series, vol. 67. Harlow: Longman Scientific & Technical; 1993.
[9] Foster I, Kesselman C. The globus project: a status report. In: Proceedings of the IPPS/SPDP '98 heterogeneous computing workshop, 1998. p. 4–18.
[10] Garbey M, Tromeur-Dervout D. A parallel adaptive coupling algorithm for systems of differential equations. J Comput Phys 2000;161(2):401–27.
[11] Garbey M, Tromeur-Dervout D. On some Aitken like acceleration of the Schwarz method. Int J Fluid Dyn 2002;40(12):1493–513.
[12] Glowinski R, Pironneau O. Numerical methods for the first biharmonic equation and for the two-dimensional Stokes problem. SIAM Rev 1979;21:167–212.
[13] Gropp W, Curfman McInnes L, Smith B. Using the scalable nonlinear equations solvers package. Technical report ANL/MCS-TM-193, Argonne National Laboratory, Argonne, IL, 1995.
[14] Lehoucq R, Sorensen DC, Yang C. ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods. Software, environments, and tools 6. SIAM; 1998.
[15] Lehoucq RB, Sorensen DC, Yang C. ARPACK users' guide: solution of large scale eigenvalue problems with implicitly restarted Arnoldi methods, 1997. http://www.caam.rice.edu/software/ARPACK/.
[16] Pernice M, Walker H. NITSOL: a Newton iterative solver for nonlinear systems. SIAM J Sci Comput 1998;19:302–18.

[17] Pickles SM, Brooke J, Costen FC, Gabriel E, Müller M, Resch M, et al. Metacomputing across intercontinental networks. Future Gener Comput Syst 2001;17(8):911–8.
[18] Rathinam M, Petzold L. Dynamic iteration using reduced order models: a method for simulation of large scale modular systems. SIAM J Numer Anal 2002;40:1446–74.
[19] Rathinam M, Petzold L. A new look at proper orthogonal decomposition. SIAM J Numer Anal 2003;41:1893–925.
[20] Smarr L, Catlett CE. Metacomputing. Commun ACM 1992;35(6):45–52.