Proceedings of ASME Turbo Expo 2011
GT2011
June 6-10, 2011, Vancouver, British Columbia, Canada

GT2011-45306

ON THE PARALLELIZATION OF A HARMONIC BALANCE COMPRESSIBLE NAVIER-STOKES SOLVER FOR WIND TURBINE AERODYNAMICS

Adrian Jackson
EPCC
James Clerk Maxwell Building
Mayfield Road
University of Edinburgh
Edinburgh EH9 3JZ, UK
adrian.jackson@ed.ac.uk

M. Sergio Campobasso*
School of Engineering
James Watt Building South
University Avenue
University of Glasgow
Glasgow G12 8QQ, UK
sergio.campobasso@glasgow.ac.uk

Mohammad H. Baba-Ahmadi
School of Engineering
James Watt Building South
University Avenue
University of Glasgow
Glasgow G12 8QQ, UK
m.baba-ahmadi@aero.gla.ac.uk

ABSTRACT

The paper discusses the parallelization of a novel explicit harmonic balance Navier-Stokes solver for wind turbine unsteady aerodynamics. For large three-dimensional problems, the use of a standard MPI parallelization based on the geometric domain decomposition of the physical domain may require an excessive degree of partitioning with respect to that needed when the same aerodynamic analysis is performed with the time-domain solver. This occurrence may penalize the parallel efficiency of the harmonic balance solver due to excessive communication among MPI processes to transfer halo data. In the case of the harmonic balance analysis, the necessity of further grid partitioning may arise because the memory requirement of each block is higher than for the time-domain analysis: it is that of the time-domain analysis multiplied by a variable proportional to the number of complex harmonics used to represent the sought periodic flow field. A hybrid multi-level parallelization paradigm for explicit harmonic balance Navier-Stokes solvers is presented, which makes use of both distributed and shared memory parallelization technologies, and removes the need for further domain decomposition with respect to the case of the time-domain analysis. The discussed parallelization approaches are tested on the multigrid harmonic balance solver being developed by the authors, considering various computational configurations for the CFD analysis of the unsteady flow field past the airfoil of a wind tubine blade in yawed wind.

INTRODUCTION

The aeromechanical design of modern large Horizontal Axis Wind Turbines (HAWT's) requires accurate analyses of complex aerodynamic features. The variability of the environmental conditions on a wide range of time-scales makes the operating conditions of HAWT's inherently unsteady. Several typical HAWT unsteady aerodynamic problems can be viewed as periodic. This is the case of stall-induced vibrations and the yawed wind regime, which occurs when the freestream wind velocity is not orthogonal to the turbine rotor. The use of the Navier-Stokes (NS) equations to study the unsteady aerodynamics associated with these operating conditions can improve the predictive accuracy of these unsteady flows with respect to lower-fidelity models, but the solution of the unsteady NS equations for complex three-dimensional 3D problems in the time-domain (TD) requires very high computational times. Fortunately, the wallclock time required by the TD NS prediction of unsteady periodic flows can be dramatically reduced by using a frequency-domain (FD) formulation and solution of the governing unsteady equations. The harmonic balance (HB) NS technology for the solution of unsteady periodic flows [1] is one of the most promising FD NS methods. The HB NS technology has been applied to the prediction of the periodic flow associated with flutter and forced response of turbomachinery blades [1–3], and various vibratory motion modes

*Address all correspondence to this author.

of aircraft configurations [4–6]. For this type of application, it has been observed that the use of the HB NS approach for the calculation of periodic flows can lead to a reduction of the wallclock time varying between one and two orders of magnitude with respect to conventional TD NS analyses. Another successful and computationally effective FD approach to the solution of unsteady periodic flows is the nonlinear frequency-domain (NLFD) method [7–9]. The NLFD technology has also been applied to the simulation of the periodic flow past rotorcraft blades [10]. Several other FD methods have been developed in the past years, among which a one-harmonic FD technique for the calculation of periodic turbomachinery flows [11], which bears some resemblance to the HB approach of hatcla02, but differs from it in that the calculation of the zeroth harmonic (mean state) is decoupled from that of the first harmonic representing the sought unsteady flow component. Numerous examples of the application of the HB and NLFD technologies to periodic flows of engineering interest exist, but a thorough review of all existing FD methods and their application is beyond the scope of this report. This paper presents and analyzes the strategies available for the parallelization of a novel HB compressible multi-block NS multigrid solver with Low-Speed Preconditiong (LSP) [12] for wind turbine unsteady aerodynamics, the development and application of which have recently been reported by the authors of this paper in the article [13].

At the code level, the main alteration one has to introduce in an existing TD NS solver to develop its HB counterpart is that an additional dimension has to be added to all arrays of the code (*e.g.*current flow solution and residual arrays), namely that for the harmonic count in the Fourier space. As a consequence, the memory requirement of the HB code grows linearly with the number of retained harmonics, and this occurrence leads to severe limitations on how HB NS parallel Message Passing Interface (MPI) solvers can be used. Such limitations are caused by the fact that in CFD codes, MPI is used in conjunction with domain decomposition. The whole computational domain is decomposed in partitions or blocks, and each core of a computer cluster node performs the operations required to update the solution of one or more blocks. At prescribed time-points (synchronization steps) all cores have to exchange flow information regarding the boundaries of the block(s) each core looks after. When using the HB NS solver, the memory needed by each block increases linearly with the number of user-requested harmonics with respect to the memory used to solve the same physical problem with the TD NS code. The HB simulation clearly requires more memory than the TD one, but the most severe problem is that the memory capacity of the computing cores may become insufficient to handle blocks which are geometrically identical to those of the TD simulation but require more memory due to the additional Fourier-space dimension. This circumstance is very likely to occur in large 3D applications since numbers of harmonics as small as 5 cause the required memory to increase by a factor of 10. Using MPI, the only solution to this problem is to further decompose the computational domain, so that the smaller geometric partitions can cope with the available memory of the cores. Each partition is now geometrically smaller, but its TD-memory requirement has to be multiplied by a factor proportional to the number of requested harmonics. Thus the overall memory demand of the geometrically smaller partitions can now cope with the memory capacity of the core. The downside of this approach is a reduction of the parallel efficiency due to the creation of additional interfaces across which the partitions have to exchange boundary informations. Furthermore, additional mantime is required when the domain-decomposition is performed manually. This paper presents an effective hybrid parallelization paradigm for HB NS solvers, which solves this problem by using a combined distributed and shared parallelization of the HB solver. This implementation allows one to avoid altering the domain decomposition for the HB NS analysis with respect to that used for the TD analysis.

After recalling the mathematical and numerical theory behind the implementation of the time- and frequency-domain multigrid (MG) NS solver reported in [13], this paper discusses several options available for the parallelization of the frequency-domain solver, and presents an optimal hybrid parallelization of the HB solver based on both distributed and shared parallel computing. Finally, the computational performance of the parallelization methods tested thus far is assessed by analyzing the relationship between computational speed and used number of cores when performing the frequency-domain analyses of the unsteady periodic flow past a two-dimensional (2D) HAWT blade section.

GOVERNING EQUATIONS

HAWT viscous flows can be computed by solving the NS equations, a system of N_{pde} nonlinear partial differential equations (PDE's) obtained by imposing the conservation of mass, momentum and energy over a control volume. For 2D laminar flows $N_{pde} = 4$ because the momentum equation has only two scalar components. Given a control volume $\mathcal{C}$ with boundary S, the Arbitrary Lagrangian-Eulerian integral form of the 2D TD NS equation is:

$$\frac{\partial}{\partial t}\left(\int_{\mathcal{C}(t)} \mathbf{U}\, d\mathcal{C}\right) + \oint_{S(t)} (\underline{\Phi}_i - \underline{\Phi}_v)\cdot d\underline{S} = 0 \qquad (1)$$

The array $\mathbf{U}$ of conservative flow variables is defined as:

$$\mathbf{U} = [\rho \;\; \rho u \;\; \rho v \;\; \rho\varepsilon]'$$

where the superscript $'$ denotes the transpose operator, and ρ, u, v and ε are respectively the flow density, the $x-$ and $y-$component

of the flow velocity vector $\underline{v}$, and the total energy per unit mass. The definition of the total energy is $\varepsilon = e + (u^2+v^2)/2$, where e denotes the internal energy per unit mass. The generalized inviscid flux vector $\underline{\Phi}_i$ is:

$$\underline{\Phi}_i = \mathbf{E}_i \underline{i} + \mathbf{F}_i \underline{j} - \underline{v}_b \mathbf{U} \tag{2}$$

where $\mathbf{E}_i$ and $\mathbf{F}_i$ are respectively the $x-$ and $y-$components of $\underline{\Phi}_i$, and they are both functions of $\mathbf{U}$. The vector $\underline{v}_b$ is the velocity of the boundary S, and the flux term $-\underline{v}_b \mathbf{U}$ is its contribution to the overall flux balance, which is nonzero only in the case of unsteady problems with moving boundaries. The generalized viscous flux vector $\underline{\Phi}_v$ is:

$$\underline{\Phi}_v = \mathbf{E}_v \underline{i} + \mathbf{F}_v \underline{j} \tag{3}$$

where $\mathbf{E}_v$ and $\mathbf{F}_v$ are respectively the $x-$ and $y-$components of $\underline{\Phi}_v$, and they are both functions of $\mathbf{U}$.

The HB formulation of the NS equations assumes that the fundamental frequency ω of the sought periodic flow field is known. The unknown periodic flow field is viewed as a truncated Fourier series in which only the first N_H harmonics are retained, and the variable N_H is a user given parameter. It has been noted that a simpler implementation of the HB NS method is obtained by reconstructing the Fourier coefficients of the truncated harmonic series representing the sought periodic solution from the knowledge of its temporal behavior at $2N_H+1$ equally spaced points over one period. Such points are defined by:

$$t_n = \frac{n}{(2N_H+1)} \frac{2\pi}{\omega}, \qquad n = 0, 1, \cdots, 2N_H \tag{4}$$

Using such TD reconstruction of the solution and the entire system of conservation laws, leads to the so called *high-dimensional harmonic balance formulation* [14] of the NS equations:

$$\omega D_c \left(\int_{C_H(t)} \mathbf{U}_H \, dC_H \right) + \oint_{S_H(t)} (\underline{\Phi}_{i,H} - \underline{\Phi}_{v,H}) \cdot d\underline{S}_H = 0 \tag{5}$$

where $\mathbf{U}_H = [\mathbf{U}(t_0)' \; \mathbf{U}(t_1)' \ldots \mathbf{U}(t_{N_H})']'$, $\Phi_{i/v,H} = [\Phi_{i/v}(t_0)' \; \Phi_{i/v}(t_1)' \ldots \Phi_{i/v}(t_{N_H})']'$, and similar expressions hold for C_H and S_H. The symbol D_c denotes a sparse block-matrix of dimension $[(2N_H+1)\times(2N_H+1)]$, all blocks of which are squared and have dimension $(N_{pde} \times N_{pde})$. Moving from the time- to the frequency-domain, the number of PDE's increases from N_{pde} to $[N_{pde} \times (2N_H+1)]$. Despite the fact that the number of PDE's to be solved has increased, the HB approach allows one to compute unsteady periodic flows at a substantially lower computational cost with respect to the time-domain approach. More details on the formulation of the TD NS equations and the complete derivation of the HB NS equations are reported in [13].

CFD SOLVER

Space discretization

The structured multi-block finite volume cell-centered parallel CFD code *COSA* [13,15,16] solves the integral form of both the TD conservation laws (system (1)) and the HB conservation laws (system (5)) making use of a second order upwind scheme. The discretization of the convective fluxes is based on Van Leer's *MUSCL* extrapolations and Roe's flux-difference splitting. Denoting by $\underline{n}$ the normal of the face of a grid cell, and dS the area of such face, the numerical approximation to the continuous convective flux component $\Phi_{i,f} = (\underline{\Phi}_i \cdot \underline{n}) dS$ through such face is:

$$\Phi^*_{i,f} = \frac{1}{2} \left[\Phi_{i,f}(\mathbf{U}_L) + \Phi_{i,f}(\mathbf{U}_R) - \frac{\partial \Phi_{i,f}}{\partial \mathbf{U}} \delta \mathbf{U} \right] \tag{6}$$

Here the superscript *, the subscript $_f$, and the subscripts $_L$ and $_R$ denote numerical approximation, face value, and value extrapolated from the left and from the right, respectively. The numerical dissipation depends on the generalized flux Jacobian $\partial \Phi_{i,f}/\partial \mathbf{U}$ and the flow state discontinuity across the cell face, defined by $\delta U = (\mathbf{U}_R - \mathbf{U}_L)$.

The discretization of the viscous fluxes is based on second order centered finite-differences. The Cartesian derivatives of the flow velocity components are computed with the chain rule, using the derivatives of such components with respect to the local generalized curvilinear coordinates associated with the grid lines, and the grid metrics.

Integration of time-domain equations

The physical time-derivative of system (1) is discretized with a second-order backward finite-difference. The set of nonlinear algebraic equations resulting from the space- and time-discretization of system (1) is then solved with an explicit approach based on the use of a fictitious time-derivative (Jameson's dual-time- stepping [17]). The discretization of the physical time- derivative of the unknown flow state by means of a second order backward finite difference, and the introduction of the derivative with respect to the fictitious time τ yield the equation:

$$V \frac{\partial \mathbf{Q}}{\partial \tau}^{n+1} + \mathbf{R}_g(\mathbf{Q}^{n+1}) = 0 \tag{7}$$

where

$$\mathbf{R}_g(\mathbf{Q}^{n+1}) = \frac{3\mathbf{Q}^{n+1} - 4\mathbf{Q}^n + \mathbf{Q}^{n-1}}{2\Delta t} V + \mathbf{R}_\Phi(\mathbf{Q}^{n+1}) \qquad (8)$$

The entries of the array $\mathbf{Q}$ are the unknown flow variables at the N_{cell} cells discretizing the computational domain. The array $\mathbf{Q}$ can be viewed as made up of N_{cell} subarrays, each of which stores the N_{pde} flow unknowns at a particular physical time. The length of $\mathbf{Q}$ is therefore $(N_{pde} \times N_{cell})$. The array $\mathbf{R}_\Phi$ stores the cell residuals, and its structure is the same as that of $\mathbf{Q}$. For each cell, the N_{pde} residuals are obtained by adding the convective fluxes $\Phi^*_{i,f}$ and the viscous fluxes $\Phi^*_{v,f}$ through all the faces of the cell. The symbol $\mathbf{R}_g$ denotes instead a residual vector which also includes the source terms associated with the discretization of physical time-derivative $\partial\mathbf{U}/\partial t$ contained in Eqn. (1). The diagonal matrix V stores the volumes of the grid cells. It can be viewed as a block-diagonal matrix of size $(N_{cell} \times N_{cell})$ with each block being the identity matrix of size $(N_{pde} \times N_{pde})$ multiplied by the volume of the cell the block refers to. Note that V is independent of the physical time-level (denoted by the superscripts $n+1$, n and $n-1$) because in this report only rigid-body grid motion is considered. The symbol Δt indicates the user-given physical time-step. Equation (7) can thus be viewed as a system of $(N_{pde} \times N_{cell})$ ordinary differential equations (ODE's) in which the unknown is $\mathbf{Q}^{n+1}$, the flow state at time-level $n+1$. The calculation of $\mathbf{Q}^{n+1}$ is performed iteratively by discretizing the fictitious time-derivative $(\partial\mathbf{Q}/\partial\tau)^{n+1}$ of Eqn. (7) with a multi-stage Runge-Kutta (RK) scheme, and marching the equations in pseudo-time until a steady state is achieved. Such steady state is the flow solution for the physical time being considered. The convergence rate is then greatly enhanced by means of local time-stepping (LTS), variable-coefficient central *implicit residual smoothing* (IRS) and a *full-approximation scheme* multigrid (MG) algorithm. Discretizing the fictitious time-derivative of Eqn. (7) with a multi-stage RK scheme yields the following iterative scheme used to obtain the solution update at each RK cycle:

$$\begin{aligned} \mathbf{W}^0 &= \mathbf{Q}_l \\ \mathbf{W}^k &= \mathbf{W}^0 - \alpha_k \Delta\tau V^{-1} L_{IRS}[\mathbf{R}_g(\mathbf{W}^{k-1}) + \mathbf{f}_{MG}] \\ \mathbf{Q}_{l+1} &= \mathbf{W}^{NS} \end{aligned} \qquad (9)$$

where k varies between 1 and the number of RK stages NS, α_k is the k^{th} RK coefficient, l is the RK cycle counter, and $\mathbf{Q}_l$ is shorthand for $\mathbf{Q}_l^{n+1}$. The symbol L_{IRS} denotes the IRS operator, and $\mathbf{f}_{MG}$ is the MG forcing function, which is nonzero when the smoother (9) is used on a coarse level after a restriction step [18]. The integration algorithm (9) takes a slightly different form when a stabilization method, used to remove the numerical instability of the MG iteration encountered in the solution of certain time-dependent problems, is adopted. Further modifications of the iterative scheme (9) are needed when using low-speed preconditioning, required for the solution of low-speed flows such as HAWT problems. Both extensions are reported in [13], and they are omitted here because their consideration does not modify the structure and the conclusions of the following discussion on the parallelization of the *COSA* solver.

Integration of harmonic balance equations

At the differential level, the only difference between system (1) and system (5) is that the physical time-derivative of the former system is replaced by a volumetric source term proportional to ω in the latter. The set of nonlinear algebraic equations resulting from the space-discretization of system (5) is thus solved with the same technique used for steady problems [15], namely the multi-stage RK smoother accelerated by LTS, IRS and MG. The introduction of the derivative with respect to the fictitious time τ yields the equation:

$$V_H \frac{\partial \mathbf{Q}_H}{\partial \tau} + \mathbf{R}_{g,H}(\mathbf{Q}_H) = 0 \qquad (10)$$

where

$$\mathbf{R}_{g,H}(\mathbf{Q}_H) = \omega V_H D_d \mathbf{Q}_H + \mathbf{R}_{\Phi,H}(\mathbf{Q}_H) \qquad (11)$$

The array $\mathbf{Q}_H$ is made up of $(2N_H+1)$ flow states referring to the physical times defined by Eqn. (4). Therefore, one has $\mathbf{Q}_H = [\mathbf{Q}'_0\ \mathbf{Q}'_1 \dots \mathbf{Q}'_{2N_H}]' = [\mathbf{Q}(t_0)'\ \mathbf{Q}(t_1)' \dots \mathbf{Q}(t_{2N_H})']'$, and each subarray of $\mathbf{Q}_H$ has length $(N_{pde} \times N_{cell})$. The arrays $\mathbf{R}_{g,H}$ and $\mathbf{R}_{\Phi,H}$ have the same structure of $\mathbf{Q}_H$. The subarray $(\mathbf{R}_\Phi)_n$ $(n = 0, 1, \cdots, 2N_H)$ denotes the grid-residuals associated with the convective and viscous fluxes at time t_n. The subarray $(\mathbf{R}_g)_n$ denotes instead a residual vector which also includes the source term $\omega V_H D \mathbf{Q}_H$. The diagonal matrix V_H is given by $V_H = I_{2N_H+1} \otimes V$. The symbol D_d denotes a sparse block-matrix of dimension $[(2N_H+1) \times (2N_H+1)]$, all blocks of which are square and have dimension $[(N_{pde} \times N_{cell}) \times (N_{pde} \times N_{cell})]$.

Equation (10) can thus be viewed as a system of $[N_{pde} \times N_{cell} \times (2N_H+1)]$ ODE's in the unknown $\mathbf{Q}_H$. The calculation of $\mathbf{Q}_H$ is performed iteratively by discretizing the fictitious time-derivative $(\partial\mathbf{Q}_H/\partial\tau)$ of Eqn. (10) with a four-stage RK scheme, and marching the equations in pseudo-time until a steady state is achieved. The IRS and the MG acceleration techniques are also used exactly as for steady and TD problems. Discretizing the fictitious time-derivative of Eqn. (10) with a multi-stage RK scheme yields the following iterative scheme used to obtain the

solution update at each RK cycle:

$$\begin{array}{ll} \mathbf{W}_H^0 & = (\mathbf{Q}_H)_l \\ \mathbf{W}_H^k & = \mathbf{W}_H^0 + -\alpha_k \Delta\tau V_H^{-1} L_{IRS,H}[\mathbf{R}_{g,H}(\mathbf{W}_H^{k-1}) + \mathbf{f}_{MG,H}] \\ (\mathbf{Q}_H)_{l+1} & = \mathbf{W}_H^{NS} \end{array} \tag{12}$$

where the HB MG forcing function is defined as $\mathbf{f}_{MG,H} = [\mathbf{f}_{MG}(t_0)'\ \mathbf{f}_{MG}(t_1)' \ldots \mathbf{f}_{MG}(t_{2N_H})']'$ with the $(2N_H+1)$ values of t_n defined by Eqn. (4), and the HB IRS operator $L_{IRS,H}$ can be viewed as a $[(2N_H+1)\times(2N_H+1)]$ block-diagonal matrix, the nonzero blocks of which are the $(2N_H+1)$ $L_{IRS}(t_n)$ operators.

The authors of this article have found that the HB integration algorithm (12) becomes numerically unstable for some time-dependent problems. More specifically, such an instability has been first encounted when trying to use the HB MG iteration (12) to compute the unsteady transonic flow past a pitching airfoil. Such transonic HB analyses could be performed only by using the stabilized HB MG iteration presented in the article [13], and are reported in the article [4]. In essence, the stabilization method of the HB MG iteration follows the steps of the stabilization of the TD MG iteration that was first proposed by Melson *et al.* [19] and consists of treating implicitly the HB source term $\omega V_H D_d \mathbf{Q}_H$ within each RK step. The introduction of this feature into the explicit integration algorithm (12) leads to a small increment of its computational cost, as the update process will now require the inversion of a $[(2N_H+1)\times(2N_H+1)]$-matrix for each grid cell. In their experience with this HB MG solver thus far, the authors of this paper have also found that the use of the stabilization process is not always essential. As an example, all analyses presented in these paper could be performed without using the aforementioned stabilization, and it was also found that the residual convergence histories of all calculations reported herein were the same with and without the stabilizing procedure. The articles [19] and [20] reported that the instability of the TD MG iteration may occur when the physical time-step Δt is significantly smaller than the pseudo-time step $\Delta\tau$. With transonic flows, for example, this may occur in the supersonic region upstream of a shock. However, the condition $\Delta\tau >> \Delta t$ does not occur for all types of fluid dynamics problems, and therefore the stabilization of the TD MG iteration is not always required. In the HB context, the equivalent physical time-step Δt is given by $\Delta t = 2\pi/\omega/(2n_H+1)$. Sinilarly to the TD case, numerical instability of the HB MH iteration is expected to occur when $\Delta\tau >> \Delta t$, but this cobdition does not occur for all fluid dynamics problems, which could explain why the stabilized TD MG iteration is not always required. The interested reader is referred to [13] for the mathematical and numerical aspects of the stabilized HB MG iteration, and for a discussion on the relationship between the stabilization and the expected trends of the residual converge history.

Other published studies have reported that, when solving the HB equations with an implicit approach, it is not possible to avoid an implicit treatment of the HB source term if the stability of the integration process is to be maintained [5, 6]. This constraint may require substantial code extensions if the HB solver is built around an existing time-domain code. It may also yield very large memory usage for storing the Jacobian associated with all $(2N_H+1)$ flow states if a Krylov-subspace method with approximate Jacobian-based preconditioning is used for the solution of the linear systems arising at each step of Newton's method. One possible solution is to use an iterative stationary linear block-solver such as block-Jacobi to solve the linear systems, as this allows one to treat separately the Jacobians associated with each flow snapshot during the integration [5]. An alternative solution to simplify the development of the HB technology around an existing implicit solver is the treatment of the HB source term presented in [21].

Finally. it should be noted that further modifications of the iterative scheme (12) are needed when using low-speed preconditioning [12, 16], required for the solution of low-speed flows such as HAWT problems. The LSP capability of the *COSA* solver has been used for all analyses reported in this article, and its formulation in the context of the stabilized HB MG iteration can be found in reference [13].

PARALLELIZATION

The HB *COSA* solver is a multi-block code, and the structure of most of its subroutines is:

do ib = 1, N_{block}
  do ih = 0, $2N_H$
    do $jcell$ = 1, $N_{cell,j}$
      do $icell$ = 1, $N_{cell,i}$

where N_{block} is the number of blocks or partitions making up the computational domain, and $N_{cell,i}$ and $N_{cell,j}$ are respectively the number of cells in the i and j directions of the current block. In the TD solver, $N_H = 0$ and therefore there is one less loop. The operations performed on a particular block are nearly entirely independent on those performed on all other blocks, and therefore each block can be processed separately and in parallel. However, the operations performed within each block, represented by the loops over harmonics and cells, depend upon other data within that block. From a parallelization viewpoint, the two innermost loops in the above description, namely those over the block cells, can be viewed as a single loop over all the cells of the block. In the case of 3D problems and data structures, the above scheme would feature three rather than two cell loops, but the parallelization would not be affected. This is because the cell cycles either loop over all the faces of the block cells (*e.g.*flux calculation), or loop directly over the cells of the block (*e.g.*solution update), and both functionalities are common to both 2D and 3D problems.

There are two main options for the parallelization of this type of code, shared or distributed memory. Distributed memory parallelization based on the message passing library MPI [22] has been the dominant method for parallelizing scientific codes for the past ten or fifteen years, primarily due to the availability of large distributed memory systems and the prohibitive cost of large shared memory systems. Shared memory parallelizations, generally undertaken using the OpenMP [23] shared memory library, have been restricted to a number of specialized high-performance computer (HPC) systems or to very small numbers of processors. However, given the current trend for multi-core processors and the fact that the number of cores in a single processor is rapidly increasing, shared memory parallelization is becoming more viable. Indeed, it is now possible to purchase systems with 48 cores that all have access to the same memory (in actual fact a 48-core shared memory system) as entry level servers, providing relatively large shared memory resources at little cost. With the processor manufacturers aiming for hundreds of cores in a single processor in the near future, shared memory systems with large numbers of cores at a relatively low cost are likely to be available soon.

For the reasons discussed above, both a shared and a distributed memory parallelization of the *COSA* code have been considered. The code parallelization is required because, although the runtime of the HB solver for any realistic 3D turbulent simulation is expected to be at least one order of magnitude smaller than that required by the TD solver, the serial execution of this type of HB analysis is still likely to require several days of computing. This is a serious limitation for the deployment of the HB technology in industrial HAWT analysis and design systems. The reduction of these runtimes necessitates the use of parallel computers. With any parallel implementation there is a trade-off between the cost of the parallelization (both in terms of the effort required to construct the parallel code and in terms of the extra runtime required for the operation of the parallel parts of the code including communications) and the performance improvements achieved by using many more processors to execute the program. For a parallel implementation to perform well, one needs to ensure that there is sufficient work for the parallel parts of the computation to do to mitigate the cost of the parallelization. From a computational viewpoint, the more work each parallel part of the program has to do the more efficient the parallelization is likely to be.

The following three subsections outline the approach to parallelizing the code using OpenMP, MPI, and a hybrid model using both OpenMP and MPI. The benefits and drawbacks of each approach are also discussed.

OpenMP parallelization

There are a number of options on how the shared memory parallelization can be undertaken. OpenMP generally exploits *loop parallelization*, taking independent iterations of loops and distributing them to a group of threads that perform these sets of independent operations in parallel. Since each of the threads can access shared data, it is generally straightforward to parallelize any loop, and no structural change to the program is required. However, this model of parallelization does impose an overhead for each loop that is parallelized: each parallelized loop requires some operations to set up the threads that will be undertaking the parallelization and the data structures they need to operate with. The performance of these operations has a detrimental effect on the runtime. However, this overhead is generally small as long as the loops that are parallelized are large and contain sufficient computational work. In this circumstance, such overhead is negligible with respect to the improved performance obtained by parallelizing the code. One additional observation is that for an OpenMP parallelization to be efficient, the number of times this overhead is incurred needs to be minimized. Therefore, in an attempt to reduce parallel efficiency losses, the computationally less intensive loops of the *COSA* solver have not been parallelized.

Given the general structure of a routine of the *COSA* code outlined in the introduction to this section, there are three options for the OpenMP parallelization: a) over the block loop, b) over the harmonic loop, and c) over the cell loop. As an overhead is incurred each time a loop is parallelized, the higher the level of the loop the parallelization is applied at, the lower the overheads will be, as discussed in [24]. Therefore, the ideal loop for an OpenMP parallelization in the typical *COSA* is the block loop.

Beside the aforementioned overheads, another major factor that can affect the performance of an OpenMP parallelization is the amount of work to be parallelized. The parallel loop should have enough iterations to enable one to use concurrently a reasonably large number of processors. If the parallel loop has a small number of steps, one can only use a small number of processors, thus restricting the potential benefits of the parallelization. Therefore, while parallelizing over the block loop is potentially the most efficient way to use OpenMP for this code, this will only be effective if there are enough blocks in the simulation to use all the available processors. This type of parallelization is efficient for complex geometries, where many blocks are required to handle the topological complexity. For the HB analysis of the flow field past simple geometries, where only a small number or even just one block is used, a block level OpenMP parallelization will provide little or no benefit to the overall runtime. In this circumstance, an OpenMP parallelization of the harmonic loop is used, and this enables one to reduce computational times even for geometrically simpler problems. While the harmonic level OpenMP parallelization is an acceptable option for simulations which use the harmonic balance method, there still remains the problem of how to use the shared memory parallelization for TD analyses of aerodynamic flows of geometrically simple domains discretized by means of a small number of blocks. In this

case, $N_H = 0$ and thus the harmonic parallelization cannot be exploited. Furthermore, for HB simulations with small numbers of blocks and using small numbers of harmonics, parallelization is also restricted. For these situations, where both the block level and the harmonic level parallelization are not appropriate, the parallelization is performed at the lowest level, namely at that of the loop over the block cells. Although this parallelization at a low loop level is not optimal, this cell parallelization is the only place where the OpenMP parallelization is applicable in situations where both the block and harmonic loops have small indices. Though not optimal, this parallel set-up will still reduce the overall runtime of the simulation.

All three OpenMP parallelizations have been implemented and each one can be selected when the code is compiled. The relative performance of the three parallelizations for a number of simulations is discussed in the results section of this paper.

MPI parallelization

Given the general code structure outlined in the introduction to this section, the only feasible target for an MPI-based distributed memory parallelization is to distribute the blocks to individual cores, with each core working on one or more blocks. The interdependence of calculations on the data within a single block makes it difficult to distribute parts of a single block across cores without having large amounts of communication to transfer the data required for the calculations among those cores. There is a long history of parallel implementations of NS solvers [25–27] using the method of distributing blocks to cores to work upon.

The MPI parallelization necessitates altering the code, adding functionality to perform parallel I/O and communicate data between blocks, that is passing changes in the data on the edge of blocks to neighboring blocks (*halo* communications). The MPI parallelization is more involved than the OpenMP parallelization, as the former requires code changes to explicitly distribute the simulation data, and perform the halo data communications. Distributing simulation blocks ensures that only a minimal amount of data needs to be communicated to undertake the calculations, but communication is nevertheless costly in terms of computational time. The benefit of the MPI over the OpenMP parallelization is that once the MPI parallel program has been set up, there are no overheads to the parallelization except for the explicit communications that have been added to the program. This is in contrast to OpenMP codes, where each parallel loop of the program has a small amount of overhead creating that parallelization. Additionally, all aspects of MPI programs are generally run in parallel, whereas only the parts of the program that the user/developer has parallelized are run in parallel in OpenMP programs. Therefore, one would expect the MPI version of *COSA* to achieve a better computational performance compared to its block OpenMP counterpart. The actual performance achieved is discussed in the results section.

The MPI parallelization is explicitly linked to the number of blocks in the simulation, so it cannot provide any benefit for simulations with small numbers of blocks. If the number of available processors is larger than the number of blocks, the given computational mesh could be subdivided into a larger number of blocks, so that all available processors can be used. Another important scenario in which further grid partitioning may be mandatory is that of large 3D simulations using a relatively large number of blocks to accomodate the geometric complexity of the problem and running an MPI CFD solver. If the multi-block grid has been generated to run a steady and/or TD analysis using all the available memory of the cluster nodes, one cannot use the same grid to perform a HB analysis on the same computer, because the blocks of this grid would have a memory requirement that is about $2N_H + 1$ times that used by the steady and TD solvers. Hence the existing multi-block grid needs further partitioning before the HB analysis can be performed.

The use of further grid partitioning has several drawbacks. One is that the overall number of halo cells would increase, and this would have a negative impact on the efficiency of the MPI code due to the increased computational burden associated with the communication of halo data among MPI processes. Additionally, further partitioning of a balanced multi-block grid (*i.e.*a grid where all the blocks have a similar number of cells) may result in difficulties encountered when simultaneously trying to keep the number of halo nodes within acceptable limits (*e.g.*by not increasing excessively the overall number of blocks), and keeping the multi-block grid balanced. This property has a strong impact on the efficiency of the MPI code, as a balanced grid results in a balanced load distribution of all MPI processes, and thus avoids any MPI process waiting in an idle state at a synchronization point until all other processes have completed their work. An additional inconvenience of further grid transformations is that these operations would require additional user time. This user time overhead could only be avoided by developing additional software, namely by coding up in a preprocessor or directly in the CFD solver the process of transforming a given structured multi-block grid into one with more and smaller blocks.

It should be emphasized that the aforementioned reduction of the computational performance associated with a high level of grid partitioning refers to explicit integration approaches, such as the multigrid iteration presented in this paper. This reduction is caused primarily by the overhead of the MPI process communications, and no significant effect of further domain decomposition on the convergence rate of the solver is experienced. In the case of many implicit solvers, an excessive degree of domain decomposition also reduces the overall computational performance of the integration. However, this happens because the preconditioner used to solve the linear systems associated with the linearizations of the NS equations becomes increasingly less effective as the block size decreases. Hence, the convergence rate

of the preconditioned linear solver decreases as the number of partitions increases. The interested reader is referred to the article [28] and the references therein for a review on the problem of the non-scalability of these preconditioners for the implicit solution of the NS equations.

The issues associated with excessive grid partitioning can be circumvented by using a *hybrid* parallelization method, as discussed below.

Hybrid parallelization

Given that there are simulation scenarios in which it may be more convenient, simpler, and efficient to keep the number of blocks within certain bounds, and these blocks can be large and therefore entail a large amount of computational work, the ideal parallelization would be able to distribute both the blocks and the work inside the blocks across processors to provide optimal performance. If simulations have fewer large blocks, however, they may struggle to fit such blocks into the memory available to a single processor. Noting that the majority of modern high-performance parallel computers are made up of a large number of individual nodes which are themselves shared-memory systems connected together, an effective solution meeting the demands of this type of large simulation would be to provide a parallelization of the program that combines parts of both the MPI and OpenMP parallelization, namely a *hybrid* parallelization. This strategy aims to combine the flexibility of the OpenMP solutions with the expected performance of the MPI solution. As the MPI parallelization distributes work over blocks, the choice of the OpenMP block parallelization is excluded for the hybrid system. So this latter uses MPI for the block distribution and either the harmonic loop or the cell loop parallelization of OpenMP.

As an example, consider the case of a HB analysis with a 10-block mesh and 10 harmonics. If further grid partitioning is to be avoided, a pure MPI simulation could use 10 cores at most. The OpenMP simulation could ideally use 10 cores if selecting the block or harmonics level loop parallelizations, and potentially more cores for the cell level parallelization. If a HPC system composed of 64 8-core nodes (giving a total of 512 cores which can be used for a MPI program) is available, the pure OpenMP parallelizations could actually use the 8 cores of a single node at most, and the pure MPI parallelization could use 10 cores from two nodes at most. A hybrid parallelization could instead use 10 nodes for the MPI parallelization and the 8 cores of each node for the OpenMP harmonic parallelization. Provided that the individual blocks require sufficient computational work to justify using 8 cores, then this method of parallelization allows for the concurrent use of 80 cores. Furthermore, each block would have access to the whole memory of the node, rather than just the memory available to one core. Therefore, the hybrid simulation allows access to almost 10 times the memory of the other types of parallelizations. This would allow for simulations with very large blocks to be undertaken without having to split those blocks down into smaller blocks to match the memory and computational resources of a particular high-performance computer.

RESULTS

The efficiency of the three OpenMP, the MPI and the hybrid MPI/OpenMP parallelizations has been assessed by computing the laminar unteady flow field past the blade section of a HAWT in yawed wind. The blade height is 45.7 m, its rotational speed ω is 17.5 RPM, the freestream wind velocity V_{fs} is 14 m/s, and a yaw angle δ of 30^o is assumed. The considered blade section is the airfoil at 90 percent blade height, which has a chord c of 3.16 m and a twist γ of 3.7^o. A kinematic model establishing the relationship between the flow conditions experienced by the section of a HAWT blade in yawed wind and the 2D harmonic motion of the airfoil in the circumferential direction has been presented in the article [13]. This model allows one to calculate the periodic flow field past the airfoil by means of a 2D simulation with harmonically moving grid, and to determine both the farfield data and the motion amplitude required by this simulation. The angular frequency of the motion equals the rotational speed of the turbine, and the reduced frequency λ can be defined as $\lambda = \omega c/W_{fs}$, where W_{fs} is the modulus of the equivalent 2D freestream velocity, namely the relative velocity vector associated with the component of the absolute farfield velocity orthogonal to the rotor plane. The relative 2D angle of attack ϕ_{fs} is obtained by subtracting the twist γ to the $inflow$ angle α_{fs}, the inclination of the equivalent 2D freestream velocity on the rotor plane. Choosing a reference temperature of 288 K, one can calculate the Mach number M_{fs} corresponding to W_{fs}. The set of input data used for the 2D unsteady moving-grid simulations of the blade sections is reported in Table 1. The airfoil selected for the blade section is the NACA0012 airfoil, and the Reynolds number has been set to 1000. The computational mesh used for all TD and HB aerodynamic analyses is depicted in Fig. 1. This C-grid has 577 points along the airfoil, 97 points in the grid cut, and 97 points in the normal-like direction. The farfield boundary is placed at about 20 chords from the airfoil, and the distance of the first grid points off the airfoil surface from the surface itself is about 0.01 % of the chord. All TD and HB analyses reported in

Table 1. INPUT PARAMETERS FOR THE 2D UNSTEADY MOVING-GRID CFD ANALYSES OF THE SECTIONS AT 90 PERCENT BLADE HEIGHT

M_{fs}	α_{fs} (^{o})	ϕ_{fs} (^{o})	h_0/c	λ
0.22	9.1	5.4	1.21	0.076

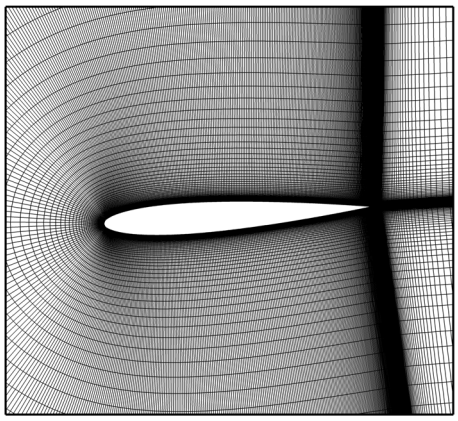

Figure 1. COMPUTATIONAL MESH.

this paper have been performed using the multigrid solver with 3 grid levels and a CFL number of 3.

The airfoil and the whole grid are inclined by the twist angle γ on the horizontal direction, and the whole grid undergoes a horizontal sinusoidal motion defined by $h_0 \sin \omega t$. The TD simulation has been performed using 128 time-intervals per period, and running the simulations for 3 periods. All HB aerodynamic analyses have been performed for N_H varying between 1 and 5. Note that the choice of a relatively thin airfoil with respect to those typically used in HAWT's, and the lack of turbulence modeling, result in this unsteady flow not being fully correspondent to those of real HAWT yawed conditions. The main objective of the following analyses, however, is to assess the efficiency of the various parallelization strategies of the HB solver rather than performing a detailed aerodynamic analysis of a particular HAWT problem. More detailed aerodynamic analyses performed with both the TD and HB solvers, and a cross comparison of the accuracy and computational performance of both methods are reported in the article [13], which highlights the suitability of the HB NS with LSP for periodic HAWT flow analyses. This is due to the fact that a relatively small number of harmonics allows one to maintain a high level of temporal accuracy, and reduce computational times by more than one order of magnitude with respect to the TD approach.

The hysteresis cycles of the lift coefficient computed by the TD simulation and the 5 HB simulations are depicted in Fig. 2, which shows that the HB analyses with $N_H \geq 2$ lead to an excellent agreement with the TD result. The convergence histories of the five HB analyses and that of the TD solver for a particular physical time are reported in Fig. 3. The variable on the x-axis is the number of multigrid iterations, and the variable l_r on the y-axis is the logarithm in base 10 of the 2-norm of the RMS of all cell-residuals for all N_{pde} equations. The HB analyses have been run until $l_r \leq 1.d-12$; the iterative solution process of each physical time-step of the TD analysis has been stopped either when $l_r \leq 1.d-12$ or after 3000 MG iterations if at this stage this convergence tolerance had not been achieved. For most physical time-steps, however, the prescribed residual tolerance of $1.d-12$ has been achieved well before the limit of 3000 MG iterations. An interesting feature is that the convergence histories of all HB analyses are practically superimposed, and thus independent of N_H. Since all HB analyses reported herein did not require the use of the stabilized integration presented in [13], the cost of a single HB MG iteration is with good approximation proportional to $2N_H+1$. The HB *speed-up* parameter, defined as the ratio of the wallclock time required to calculate three periods with the TD solver and a single period with the HB solver for each of the adopted five values of N_H is reported in Table 2. The first row of speed-up parameters refers to results computed using the aforementioned residual tolerance l_r of $1.d-12$, and it shows that the accurate HB solution obtained with $N_H = 3$ can be obtained 15 times faster than with the TD analysis reported herein. The blade forces, however, may achieve an acceptable level of convergence with less stringent residual tolerances. Indeed, comparing the results of the TD simulation with $l_r = 1.d-12$ and that with $l_r = 1.d-09$ reveals that the maximum difference of the lift and drag coefficients with respect to their averages over the third period computed with $l_r = 1.d-12$ is smaller than $1.d-01\%$. Similarly, comparing the results of the HB simulations with $l_r = 1.d-12$ and that with $l_r = 1.d-09$ reveals that the maximum difference of the lift and drag coefficients with respect to their averages over the third period computed with $l_r = 1.d-12$ is of order $1.d-02\%$. The second row of speed-up parameters of table 2 refers to results computed using residual tolerance l_r of $1.d-09$, and it shows that the HB solution obtained with $N_H = 3$ can be obtained 8 times faster than with the TD analysis.

Table 2. ACCELERATION FACTORS OF HB ANALYSES WITH RESPECT TO TIME-DOMAIN ANALYSIS FOR THE 90 % BLADE SECTION

l_r	N_H	1	2	3	4	5
1.d-12	speed-up	34.6	20.7	14.8	11.5	9.4
1.d-09	speed-up	19.0	11.2	8.1	6.2	5.2

The Mach number contours for the position in which the airfoil has maximum speed and moves to the right ($\cos \omega t = 1$) are presented in Fig. 4. For the same position and velocity, Fig. 5 depicts the percentage difference E of local Mach num-

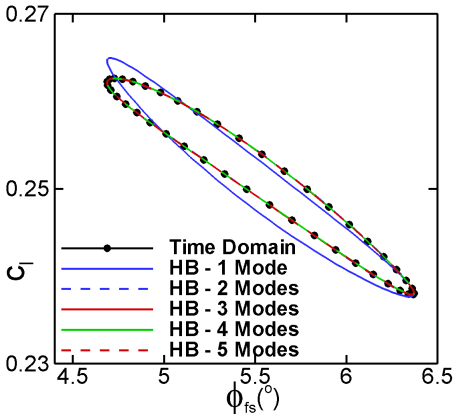

Figure 2. LIFT COEFFICIENT HYSTERIS LOOPS OF 90 % BLADE SECTION COMPUTED WITH TD AND FIVE HB ANALYSES.

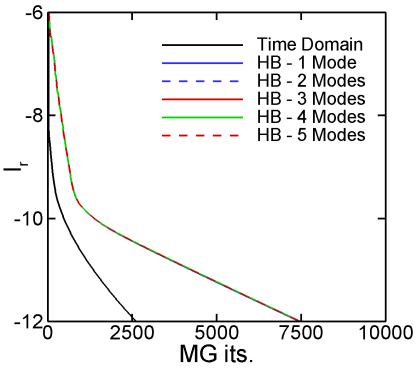

Figure 3. CONVERGENCE HISTORIES OF TD AND HB ANALYSES FOR 90 % BLADE SECTION

ber betwee the result of the TD simulation and that of the HB simulation with $N_H = 3$. The definition of this variable is $E = [(M_{TD} - M_{HB3})/M_{fs}] \times 100$, and Fig. 5 highlights that the local error E of the HB simulation with 3 harmonics relative to the TD result is everywhere smaller than 0.06 percent.

The parallel efficiency of the OpenMP block, harmonic and cell parallelizations, and the MPI parallelization have been assessed on three different clusters. The nodes of the first cluster (MATRIX) have 8 cores and are 2-way quad-core Opteron 2.1GHz with 32 GB of RAM. The second cluster is a Cray XT6 with 24-core nodes, each made up of 2-way 12-core Opteron 2.1GHz Magny Cours processors with 32GB of memory. The third cluster is a Bull machine with 8-core nodes, each consisting of 2-way quad-core Intel Xeon X5570 (Nehalem-EP) 2.93 GHz processors with 24 GB of memory. Two 1-block meshes have been used to assess the parallel efficiency of the OpenMP

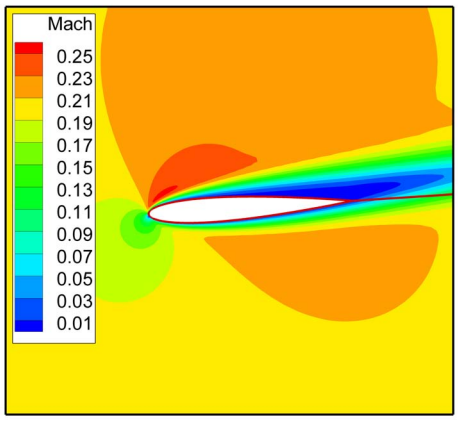

Figure 4. MACH CONTOURS FOR MAXIMUM AIRFOIL SPEED ($\cos\omega t = 1$).

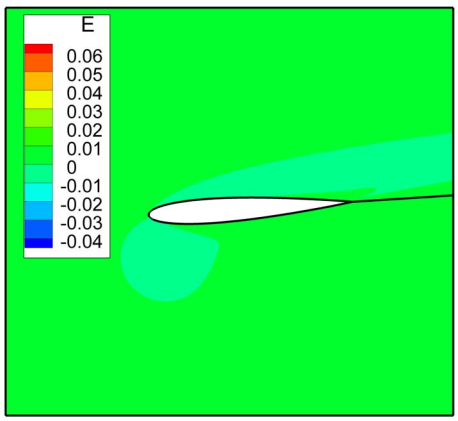

Figure 5. CONTOURS OF MACH NUMBER ERROR OF HB 3 WITH RESPECT TO TD RESULT FOR MAXIMUM AIRFOIL SPEED ($\cos\omega t = 1$).

harmonic and cell parallelizations. One is the (769×97) grid described earlier in this section; the other is a (2305×289) C-grid with 1729 points along the airfoil, 289 points in the grid cut, and 289 points in the normal-like direction. The farfield boundary is placed at about 20 chords from the airfoil, and the distance of the first grid points off the airfoil surface from the surface itself is about 0.007 % of the chord. In the rest of this section, the former grid is referred to as 'coarse grid', and the latter one

as 'fine grid'. Multi-block grids are required for testing the efficiency of the OpenMP and MPI block parallelizations. To this end, both the coarse and the fine 1-block grids have been transformed into 32-block counterparts. Both multi-block grids have been obtained by defining 16 blocks in the circumferential direction and 2 blocks in the normal-like direction. All blocks of the coarse multi-block grid have the same size, namely 49 points in the circumferential direction and 49 points in the normal-like direction. All blocks of the fine multi-block grid also have the same size, namely 144 points in the circumferential direction and 144 points in the normal-like direction. This partitioning pattern made up of 'square blocks' results in a minimal number of halo cells. This feature contributes to the reduction of the overhead of the MPI code, as it minimizes the amount of halo data to be exchanged among MPI processes.

In order to measure the parallel efficiency of the OpenMP block, harmonic and cell parallelizations, a speed-up factor defined as the ratio of the wallclock time taken by the HB analysis using one thread and the wallclock time taken by the same HB analysis using a given number of threads has been used. All OpenMP HB analyses use 8 harmonics. The coarse-grid calculations are run for 500 multigrid cycles, and the fine-grid calculations are run for 50 multigrid cycles. The low-speed preconditioner is used for all calculations reported in this paper. The parallel efficiency of the OpenMP harmonic and cell parallelizations as functions of the number of threads is reported in Figures 6 and 7 respectively. Both the coarse and fine 1-block grids have been used for these simulations, and the same calculations have been performed on all three available computer clusters. The first part of each entry of the legends provides the number of cores and the type of the processor of the nodes of the cluster the analyses have been run on. The second part of the entry identifies the refinement of the adopted grid: the symbols 'cs-g' and 'fn-g' denote coarse and fine grid respectively. As expected, the speed-up grows in a sublinear fashion with the number of threads in all cases, giving values of about 2.5 for calculations using 8 threads. The efficiency of the harmonic and cell parallelization appears to be quite similar, in contrast to the predictions made in the preceding sections. One of the possible reasons for this could be that the strucure of a few routines does not conform to the general scheme provided in the section 'parallelization', as the harmonic and cell loops are swapped. Such an inversion results in a penalization on the parallel efficiency of the solver based on the harmonic parallelization. It should also be noted that some of the routines have deliberately not been parallelized, and this has been made in the attempt to reduce the OpenMP parallelization overheads previously discussed. Further investigations are presently underway to verify whether the parallelization of part of the remaining serial routines may lead to an overall positive effect on the parallel efficiency of the OpenMP parallelizations.

The parallel efficiency of the OpenMP block, harmonic, and cell parallelizations for the coarse and fine 32-block analyses

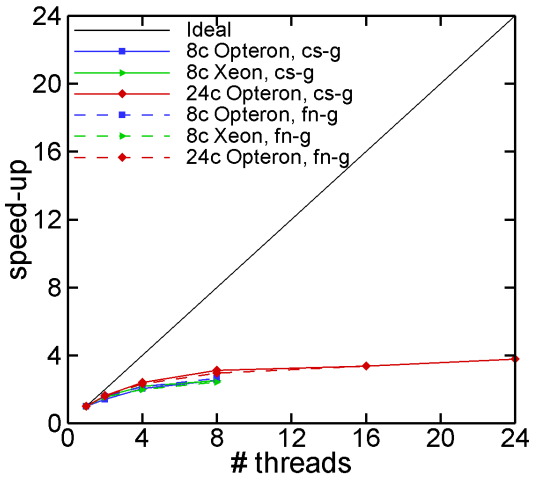

Figure 6. SPEED-UP FACTORS OF HB8 ANALYSIS USING 1-BLOCK COARSE AND FINE GRIDS ACHIEVED WITH OPENMP HARMONIC PARALLELIZATION.

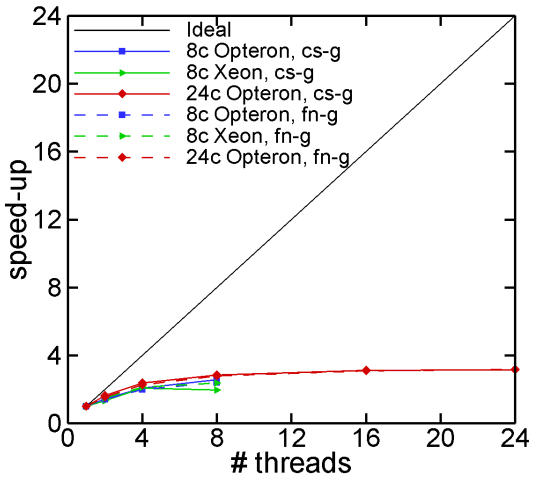

Figure 7. SPEED-UP FACTORS OF HB8 ANALYSIS USING 1-BLOCK COARSE AND FINE GRIDS ACHIEVED WITH OPENMP CELL PARALLELIZATION.

with $N_H = 8$ are reported in Figures 8, 9 and 10 respectively. The same calculations have been performed on all three available clusters. Also in this case, the speed-up of all three OpenMP parallelizations varies in a sublinear fashion. but the parallel efficiency of the higher-level block parallelization is higher than that of the lower-level harmonic and cell parallelizations. In fact, the efficiency of the harmonic and cell parallelization using 8 threads is about 2.5, whereas that of the block parallelization

using the same number of threads is about 4. An interesting phenomenon is clearly highlighted by the data referring to the Cray cluster featuring the 24-core Opteron for calculations with more than 8 threads. Figures 9 and 10 show that the efficiency of the OpenMP harmonic and loop parallelizations is always higher when using the fine rather than the coarse grid, and the difference between the two efficiencies rapidly grows as the number of threads goes to 24. In the case of the OpenMP harmonic parallelization, this happens because the number of parallel loops is the same in the coarse and fine grid calculations, but the amount of work of each step of such parallel loops is bigger when using the fine grid. Therefore, a comparable parallel overhead associated with setting up the parallel loops has a smaller impact in the case of the fine grid calculation. The same mechanism takes place in the case of the OpenMP loop analyses, though in this case one also has more parallel loops with both the coarse- and fine-grid calculations, and this effect tends to lower the parallel efficiency of both sets of calculations. Nevertheless, the effect associated with the greater amount of work at each step of the parallel loops when using the fine grid prevails, and therefore the parallel efficiency of the fine-grid calculations remains significantly higher than that of their coarse-grid counterpats when using the OpenMP loop parallelization. The results presented thus far also show that the performance of the harmonic and lopp parallelizations is worse in the case of the multi-block grids. This is due to the fact that, when using either parallelization with a multi-block grid, the number of parallel loops is higher and the amount of work of each parallel loop is smaller than with a single-block grid. Both occurrences significantly reduce the parallel efficiency of the multi-block with respect to that of the single-block grid analysis.

In the case of the MPI parallelization, the definition of the speed-up parameter is the same as for the OpenMP case, except for the fact that threads are replaced by processes. One structural difference between the MPI and the OpenMP block parallelization is that an MPI process can handle more than one block, whereas an OpenMP thread always looks after the operations associated with a single block. The HB analyses with 8 harmonics using the coarse and fine 32-block grids have been run on all three available clusters. The coarse grid amalysis has used 1000 MG cycles and the fine grid analysis has used 100 MG cycles. The speed-up factors of the MPI HB analyses are reported in Fig. 11. Also in the case of the MPI parallelization, the speed-up grows in a sublinear fashion with the number of processes. As expected, however, the efficiency of the MPI parallelization is substantially higher than that associated with the OpenMP parallelizations. The highest speed-up is obtained on the MATRIX cluster, where the speed-up observed when using 32 cores is about 24 with both the coarse- and the fine-grid analysis.

The parallel efficiency of the hybrid MPI/OpenMP HB solver has been assessed by using the 32-block fine grid, choosing 8 harmonics to represent the sought periodic flow field and performing 100 MG iterations. The speed-up factor has been defined as the ratio of the wallclock time taken by the HB analysis using one cluster node and the wallclock time taken by the same HB analysis using a given number of nodes. The performance assessment of the hybrid code has been performed on the Cray and Bull clusters. In the case of the 8-core processor, one MPI process per node and eight OpenMP threads per process (handling

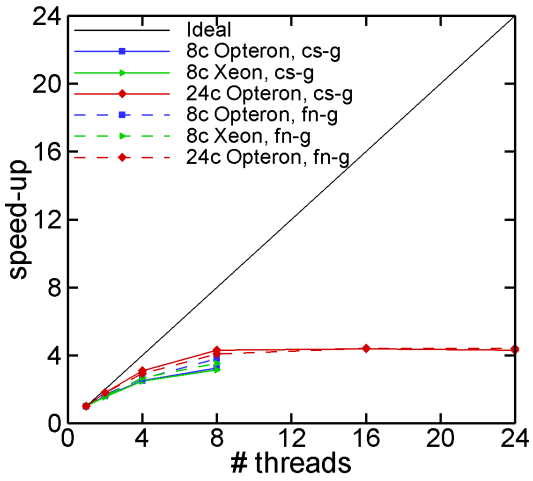

Figure 8. SPEED-UP FACTORS OF HB8 ANALYSIS USING 32-BLOCK COARSE AND FINE GRIDS ACHIEVED WITH OPENMP BLOCK PARALLELIZATION.

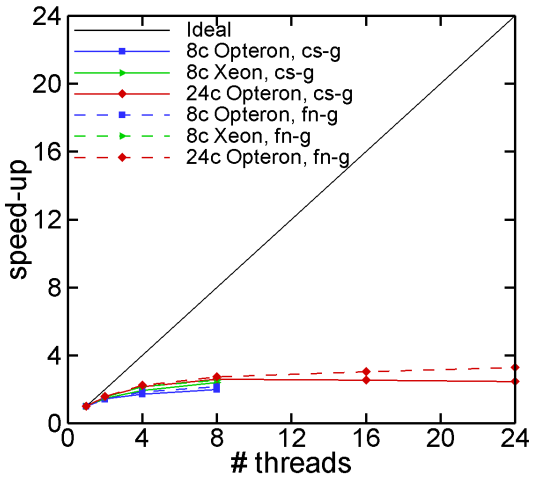

Figure 9. SPEED-UP FACTORS OF HB8 ANALYSIS USING 32-BLOCK COARSE AND FINE GRIDS ACHIEVED WITH OPENMP HARMONIC PARALLELIZATION.

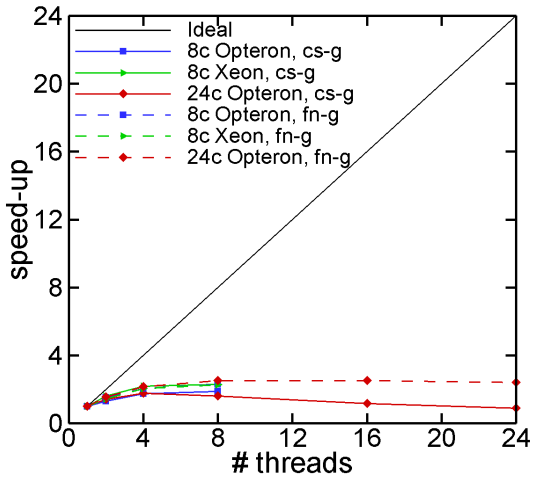

Figure 10. SPEED-UP FACTORS OF HB8 ANALYSIS USING 32-BLOCK COARSE AND FINE GRIDS ACHIEVED WITH OPENMP CELL PARALLELIZATION.

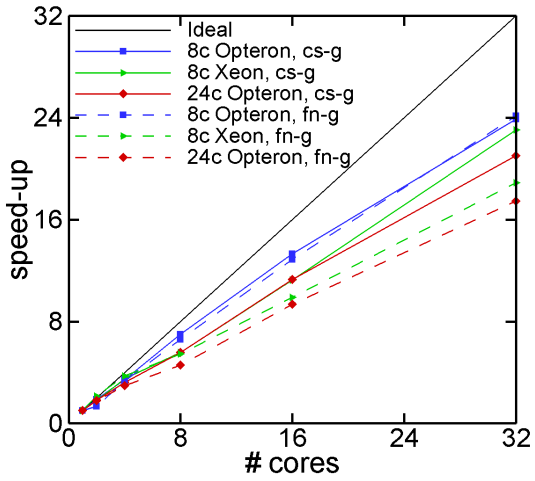

Figure 11. SPEED-UP FACTORS OF HB8 ANALYSIS USING 32-BLOCK COARSE AND FINE GRIDS ACHIEVED WITH MPI PARALLELIZATION.

the computational work associated with the flow harmonics of their MPI process) are used for all calculations. In the case of the 24-core processor, four MPI processes per node and six OpenMP threads per process (handling the computational work associated with the flow harmonics of their MPI process) are used for all calculations. The parallel efficiency of the hybrid MPI/OpenMP parallelizations as functions of the number of nodes is reported in Figures 12. The first part of each entry of the legends provides the number of cores and the type of the processor of the nodes of the cluster the analyses have been run on. The second part of the entry identifies the type of the adopted OpenMP parallelization: the symbols 'hyb-lp' and 'hyb-nh' denote loop and harmonic parallelization respectively. These results highlight a generally good scalability of the hybrid parallelizations of the HB code. The best efficiency is observed on the Bull cluster, where the parallel efficiency of the 32-node analyses varies between 28 and 30.

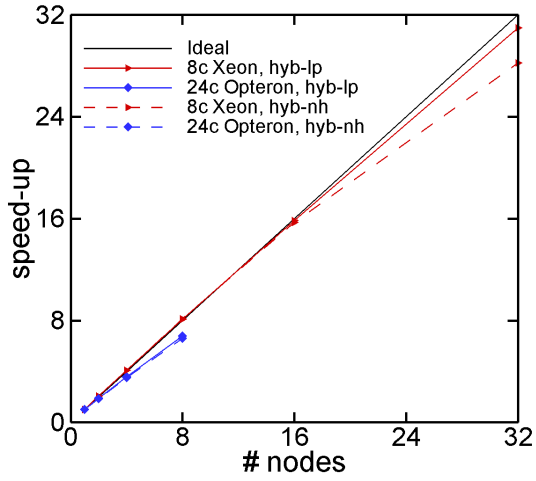

Figure 12. SPEED-UP FACTORS OF HB8 ANALYSIS USING 32-BLOCK FINE GRID ACHIEVED WITH HYBRID MPI/OPENMP PARALLELIZATION.

CONCLUSIONS

The analysis of HAWT periodic flows by means of a harmonic balance Navier-Stokes solver enhanced with low-speed preconditioning can reduce runtimes by more than one order of magnitude with respect to the corresponding time-domain analysis. Despite such reduction, wallclock times required for the HB aerodynamic analysis of 3D turbulent HAWT flows will remain high. For this reason, one fundamental means of accelerating the diffusion of this emerging technology is to use advanced parallel computing strategies. The fact that the memory requirement of the HB solver increases linearly with $(2N_H + 1)$ with respect to the memory requirement of the conventional time-domain solver, with N_H being the number of retained complex harmonics, yields substantially higher memory demands of the HB code with respect to its TD counterpart. Given the usual structure of modern parallel machines, consisting of interlinked computational nodes, each of which is essentially a multi-core distributed memory unit, the aforementioned enhanced memory requirement of the HB

solver poses grid partitioning constraints for its use in parallel. This paper has discussed and demonstrated a hybrid distributed (MPI)/ shared (OpenMP) parallelization strategy for the parallelization of explicit multigrid HB solvers which will allow an optimal exploitation of modern parallel computers for HB NS HAWT aerodynamic analyses, thus boosting the deployment of this CFD technology in present and future design practice.

The underlying MPI and OpenMP parallelizations have also been discussed and their performance analyzed. Though not thoroughly reported in this paper, the tests conducted with the prototype versions of the presented parallelizations of the HB COSA solver have highlighted that the parallel efficiency of the underlying MPI and OpenMP parallelizations can vary significantly depending on the chosen compiler, the processor type, and the hardware/software used to connect the cluster nodes. These issues are beyond the scope of this paper, and will be reported in forthcoming publications along with the optimization of the individual MPI and OpenMP parallelizations.

ACKNOWLEDGMENT

This work has been supported by the Engineering and Physical Sciences Research Council under grant EP/F038542/1. Part of the simulations reported herein were performed on the MATRIX cluster at the Consorzio Interuniversitario per le Applicazioni di Supercalcolo di Universita' e Ricerca (CASPUR), which is hereby acknowledged. EPCC is also kindly acknowledged for its support on parallel computing matters.

REFERENCES

[1] Hall, K., Thomas, J., and Clark, W., 2002. "Computations of unsteady nonlinear flows in cascades using a harmonic balance technique". *AIAA Journal,* ***40***(5), May, pp. 879–886.

[2] Su, X., and Yuan, X., 2010. "Implicit Solution of Time-Spectral Method for Periodic Unsteady Flows". *International Journal for Numerical Methods in Fluids,* ***63***(7), pp. 860–876.

[3] van der Weide, E., Gopinath, A., and Jameson, A., 2005. Turbomachinery Applications with the Time Spectral Method. AIAA paper 2005-4905, June. 17th AIAA Computational Fluid Dynamics Conference, Torontp, Ontario, Canada.

[4] Da Ronch, A., Choreyshi, M., Badcock, K., Goertz, S., Widhalm, M., Dwight, R., and Campobasso, M., 2010. Linear Frequency Domain and Harmonic Balance Predictions of Dynamic Derivatives. AIAA paper 2010-4699, July. 28th AIAA Applied Aerodynamics Conference, Chicago, Illinois.

[5] Sicot, F., Puigt, G., and Montagnac, M., 2008. "Block-Jacobi Implicit Algorithms for the Time Spectral Method". *AIAA Journal,* ***46***(12), December, pp. 3080–3089.

[6] Woodgate, M. A., and Badcock, K. J., 2009. "Implicit Harmonic Balance Solver for Transonic Flows with Forced Motions". *AIAA Journal,* ***47***(4), April, pp. 893–901.

[7] McMullen, M., and Jameson, A., 2006. "The computational efficiency of non-linear frequency domain methods". *Journal of Computational Physics,* ***212***(2), pp. 637–661.

[8] McMullen, M., Jameson, A., and Alonso, J., 2002. Application of a Non-linear Frequency-domain Solver to the Euler and Navier-Stokes Equations. AIAA paper 2002-0120, January. 40th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada.

[9] McMullen, M., Jameson, A., and Alonso, J., 2001. Acceleration of convergence to a periodic steady state in turbomachinery flows. AIAA paper 2001-0152, January. 39th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada.

[10] Kumar, M., and Murthy, V., 2006. Rotor Blade Response Based on CFD in the Frequency Domain. AIAA paper 2006-438, January. 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada.

[11] He, L., 2008. "Harmonic Solution of Unsteady Flow Around Blades with Separation". *AIAA Journal,* ***46***(6), June, pp. 1299–1307.

[12] Venkateswaran, S., and Merkle, C., 1999. "Analysis of preconditioning methods for the Euler and Navier-Stokes equations". 30^{th} *VKI Lecture Series on Computational Fluid Dynamics.*

[13] Campobasso, M., and Baba-Ahmadi, M., 2011. Analysis of Unsteady Flows Past Horizontal Axis Wind Turbine Airfoils Based on Harmonic Balance Compressible Navier-Stokes Equations with Low-Speed Preconditioning. ASME paper GT2011-45303.

[14] Liu, L., Thomas, J., Dowell, E., Attar, P., and Hall, K., 2006. "A comparison of classical and high dimensional harmonic balance approaches for a Duffing oscillator". *Journal of Computational Physics,* ***215***(1), pp. 298–320.

[15] Campobasso, M., and Baba-Ahmadi, M., 2010. "Ad-hoc Boundary Conditions for CFD Analyses of Turbomachinery Problems with Strong Radial Flow Gradients at Farfield Boundaries". *ASME paper GT2010-22176, to appear in the Journal of Turbomachinery.*

[16] Campobasso, M., Bonfiglioli, A., and Baba-Ahmadi, M., 2009. "Development of Efficient and Accurate CFD Technologies for Wind Turbine Unsteady Aerodynamics". In Proceedings of the Conference on Modeling Fluid Flow, J. Vad, ed., Vol. **2** of *14th Event of International Conference Series on Fluid Flow Technologies held in Budapest,* Department of Fluid Mechanics, Budapest University of Technology and Economics, pp. 879–886.

[17] Jameson, A., 1991. Time Dependent Calculations Using

Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings. AIAA paper 91-1596.

[18] Briggs, W., Henson, V., and McCormick, S., 2000. *A Multigrid Tutorial, Second edition*. SIAM, Philadelphia, USA.

[19] Melson, N., Sanetrik, D., and Atkins, H., 1993. "Time-accurate Navier-Stokes calculations with multigrid acceleration". *Proc. 6^{th} Copper mountain Conference on Multigrid Methods*, pp. II423–II437.

[20] Arnone, A., Liou, M.-S., and Povinelli, L., 1993. Multigrid Time-Accurate Integration of Navier-Stokes Equations. Technical Memorandum NASA TM 106373, ICOMP-93-37, Lewis Research Center, Cleveland, OH, USA, November.

[21] Thomas, J., Duster, C., Dowell, E., and Hall, K., 2009. Unsteady Flow Computation Using a Harmonic Balance Approach Implemented About the OVERFLOW 2 Flow Solver. AIAA paper 2009-4270, June. 19th AIAA Computational Fluid Dynamics Conference, San Antonio, Texas.

[22] Forum, M. MPI: A message-passing interface standard. available at: `http://www.mpi-forum.org`.

[23] Forum, M. Openmp architecture review board. openmp fortran application program interface, version 1.1. available from: `http://www.openmp.org`.

[24] Johnson, S., Leggett, P., Ierotheou, C., Spiegel, A., an Mey, D., and Hoerschler, I., 2008. "Nested parallelization of the flow solver TFS using the ParaWise parallelization environment". In *OpenMP Shared Memory Parallel Programming*, M. Mueller, B. Chapman, B. de Supinski, A. Malony, and M. Voss, eds., Vol. 4315 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, pp. 217–229.

[25] Mulas, M., Beeri, Z., Golby, D., Surridge, M., and Talice, M., 1997. "A parallel Navier-Stokes code for large industrial flow simulations". In *Fifteenth International Conference on Numerical Methods in Fluid Dynamics*, P. Kutler, J. Flores, and J.-J. Chattot, eds., Vol. 490 of *Lecture Notes in Physics*. Springer Berlin / Heidelberg, pp. 450–455.

[26] Issman, E., Degrez, G., and De Keyser, J., 1994. "A parallel multiblock Euler/Navier-Stokes solver on a cluster of workstations using pvm". In *High-Performance Computing and Networking*, W. Gentzsch and U. Harms, eds., Vol. 796 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, pp. 157–162.

[27] Elman, H., Howle, V. E., Shadid, J., and Tuminaro, R., 2003. "A parallel block multi-level preconditioner for the 3D incompressible Navier-Stokes equations". *J. Comput. Phys,* ***187***, pp. 504–523.

[28] Benzi, M., 2002. "Preconditioning Techniques for Large Linear Systems: A Survey". *Journal of Computational Physics,* ***182***, pp. 418–477.