Journal of Flow Visualization & Image Processing, vol. 14, pp. 267–286, 2007

A PARALLEL COMPUTATIONAL CODE FOR THE PROPER ORTHOGONAL DECOMPOSITION OF TURBULENT FLOWS

Giancarlo Alfonsi[*]
Dipartimento di Difesa del Suolo, Università della Calabria Via P. Bucci 42b, 87036 Rende (Cosenza), Italy

Leonardo Primavera
Dipartimento di Fisica, Università della Calabria, Via P. Bucci 33b, 87036 Rende (Cosenza), Italy

A parallel computational code for the execution of the Proper Orthogonal Decomposition (POD) of turbulent flow fields is developed. The POD is an analytically founded statistical technique that allows the extraction of appropriately defined modes of the flow from the background field, allowing the determination of the coherent structures of turbulence. The technique operates through the selection of a basis for a modal decomposition of a set of signals, to be processed with the use of the spectral theory of compact self-adjoint operators. Details are given on computational aspects with regard to the different phases of the numerical method and the development of the related parallel code is described. Computational tests corresponding to different domains and number of processors are executed on a HP 9000-V2500 parallel computer and the results are shown in terms of parallel performance of the different phases of the calculations separately considered, and of the computational code as a whole.

1. INTRODUCTION

In turbulence research, a wide class of methods of investigation involves numerical simulations. Numerical simulation of turbulence implies the execution of the numerical integration of the three-component unsteady Navier–Stokes equations on an appropriate computing domain, for an adequate number of time steps. Different numerical techniques, ranging from finite differences, finite elements, spectral methods and appropriate combinations of the basic algorithms in mixed techniques, can be used [1, 2]. One of the problems involved in this activity is the difference that may exist between a solution of the Navier–Stokes equations as an exercise of numerical analysis — whatever complex it may result — and a solution of the same equations with the aim of obtaining a precise correlation of the results with turbulence physics. In the latter case, the accuracy of the calculations has to be deeply monitored and the governing equations have eventually to be further manipulated in following one of the existing approaches to the numerical simulation and/or modeling of turbulence. Three are the main approaches to the numerical simulation and

[*]Address all correspondence to Dr. G. Alfonsi. alfonsi@dds.unical.it

modeling of turbulent flows: RANS (Reynolds-Averaged Navier–Stokes equations), LES (Large Eddy Simulation), and DNS (Direct Numerical Simulation of turbulence). Review works on the three techniques are due to Speziale [3], Lesieur and Métais [4], Moin and Mahesh [5], respectively. The major difficulty in performing turbulence calculations of practical interest lies in the remarkable amount of computational resources required. The consequence of this fact has been that, for a long time, only rather simple flow cases have been numerically analyzed.

The advent of supercomputing technologies has completely changed this scenario, opening new perspectives in the field of the high-performance computational fluid dynamics [6, 7 and references therein]. Modern supercomputing techniques have the potential of greatly increasing the amount of information gathered during a research of computational nature [8]. Moreover, the continuous effort in studying turbulence in its full complexity (three-dimensionality and unsteadiness) has brought researchers to manage large amount of data. A typical turbulent-flow database includes all three components of the fluid velocity at all points of a three-dimensional domain, gathered for an adequate number of time steps of the turbulent statistically steady state. Such databases contain much information about the physics of a turbulent flow but, in the formation of each variable, all turbulent scales have contributed and the effect of each scale is nonlinearly combined with that of all others. In turn, it is recognized that not all scales contribute to the same degree in determining the physical properties of a turbulent flow. Methods have been devised to extract from a turbulent-flow database only the information relevant for understanding of the physical properties of the flow. This implies the separation of appropriately defined modes of the flow from the background field or, equivalently, the eduction of the coherent structures of the flow. There are several techniques for extraction of the coherent structures of turbulence [9–11]. Among them, a powerful technique is the Proper Orthogonal Decomposition (POD).

In this work, a parallel computational code for the execution of the POD is developed. The computational procedure is highly composite, in the sense that it involves several groups of mathematical operations of different nature (Sections 2 and 3). With reference to the computational and parallel issues involved in the procedure, Lang [12] describes two techniques for speeding-up eigenvalue and singular value computations on shared memory parallel computers. Suda et al. [13] propose a new parallelization scheme for the Hessenberg double shift QR algorithm and demonstrate an implementation of the method on a Fujitsu AP1000 multicomputer system. Reeve and Heath [14] report on the parallelization of a code for reduction of a real symmetric matrix to the tridiagonal form, based on the Householder method and the QL algorithm for its diagonalization. Results of the tests performed on a sixteen-node IBM SP2 machine are reported.

The present work is organized as follows. In Section 2, the POD technique is outlined. In Sections 3 and 4, algorithm and memory allocation are analyzed. In Section 5, the parallelization strategy is presented. Section 6 contains a description of the HP 9000-V2500. The results of the performance tests are presented in Section 7 and concluding remarks are given at the end.

2. PROPER ORTHOGONAL DECOMPOSITION

The Proper Orthogonal Decomposition is an analytically founded statistical technique that can be applied for the extraction of the coherent structures of turbulence. The method has been first introduced in turbulent flow analysis by Lumley [15] and is extensively presented in [16, 17].

By considering a set of temporal realizations of a nonhomogeneous, square integrable, three-component, real-valued velocity field $v_i(x_j, t)$ on a finite domain D (i, $j = 1, 2, 3$), one wants to find the most similar function to the elements of the set on average, i.e., to determine the highest mean-square correlated structure with all the elements $v_i(x_j, t)$ of the set. This corresponds to determination of a deterministic vector function $\phi_i(x_j)$, so that:

$$\max_{\psi} \frac{\left\langle \left| \left(v_i\left(x_j, t\right), \psi_i\left(x_j\right) \right) \right|^2 \right\rangle}{\left(\psi_i\left(x_j\right), \psi_i\left(x_j\right) \right)} = \frac{\left\langle \left| \left(v_i\left(x_j, t\right), \phi_i\left(x_j\right) \right) \right|^2 \right\rangle}{\left(\phi_i\left(x_j\right), \phi_i\left(x_j\right) \right)} \quad . \tag{1}$$

[(a, b) is the inner product, $\langle\ \rangle$ is the average] or, equivalently, determination of the member ($\phi_i(x_i)$) that maximizes the normalized inner product of the candidate structure ($\psi_i(x_j)$) with the field $v_i(x_j, t)$. A necessary condition for problem (1) is that $\phi_i(x_j)$ is an eigenfunction, solution of the eigenvalue problem, and the Fredholm integral equation of the first kind:

$$\int_D R_{ij}\left(x_l, x'_k\right) \phi_j(x'_k) dx'_k = \int_D \left\langle v_i\left(x_k, t\right) v_j\left(x'_k, t\right) \right\rangle \phi_j\left(x'_k\right) dx'_k = \lambda \phi_i(x_l)\,, \tag{2}$$

where $R_{ij} = \langle v_i(x_l, t) v_j(x'_k, t) \rangle$ is the two-point velocity correlation tensor. When D is bounded, there exists a denumerable infinity of solutions of (2) (Hilbert–Schmidt theory) and these solutions are called the empirical eigenfunctions $\phi_i^{(n)}(x_j)$ (normalized, $\|\phi_i^{(n)}(x_j)\| = 1$). To each eigenfunction there corresponds a real positive eigenvalue $\lambda^{(n)}$ (R_{ij} is non-negative by construction) and the eigenfunctions form a complete set. Every member of the original set can be reconstructed by means of a modal decomposition in the eigenfunctions themselves:

$$v_i(x_j, t) = \sum_n a_n(t) \phi_i^{(n)}(x_j) \tag{3}$$

which can be seen as a decomposition of the originary random field into deterministic structures $\phi_i^{(n)}(x_j)$, with random coefficients. The modal amplitudes are uncorrelated and their mean-square values are the eigenvalues themselves:

$$\left\langle a_n(t)a_m(t)\right\rangle = \delta_{nm}\lambda^{(n)}, \tag{4}$$

with δ_{nm} being the Kroneker's delta. A diagonal decomposition of R_{ij} holds:

$$R_{ij}(x_l, x'_k) = \sum_n \lambda^{(n)} \phi_i^{(n)}(x_l)\phi_j^{(n)}(x'_k), \tag{5}$$

and this implies that the contribution of each different structure (eigenfunction) to the average content of turbulent kinetic energy of the flow can be separately calculated as follows:

$$E = \int_D \left\langle v_i\left(x_j, t\right) v_i\left(x_j, t\right)\right\rangle dx_j = \sum_n \lambda^{(n)}, \tag{6}$$

where E is the total turbulent kinetic energy content in the domain D.

The POD has been used in Rayleigh–Benard turbulent convection problems by Sirovich and Park [18], Park and Sirovich [19], Sirovich and Deane [20], in free shear-flow problems by Sirovich et al. [21], Kirby et al. [22], Arndt et al. [23], and Gordeyev and Thomas [24], and in wall bounded turbulent flow problems by Aubry et al. [25], Moin and Moser [26], Sirovich et al. [27], Ball et al. [28], and Webber et al. [29]. It has to be noted that, in all the aforementioned works, flow cases with limited sizes and specific symmetries have been considered, and no supercomputational tools have been used. A typical example is that of the flow in a plane channel, a problem characterized by two homogeneous directions (streamwise and spanwise). In this case, the correlation tensor has to be evaluated — for all the three velocity components — only along the nonhomogeneous direction (that orthogonal to the walls). Fully nonhomogeneous problems (the great majority in practical applications) require remarkably higher computational resources and, thus, supercomputing technologies.

In this work, a parallel code for the POD is developed. The code is general, it handles three-component velocity fields, associated to three-dimensional domains. It can be used in all kind of problems of practical interest, since it does not require any particular symmetry or regularity. The two-point velocity correlation tensor on the left hand-side of (2) is calculated in its complete form, so that the optimal representation of the velocity field outlined above is computed along all the three directions x, y, z.

3. THE POD ALGORITHM

The POD algorithm includes the following main phases (besides additional operations of minor relevance).

- Phase CORR. Evaluation of the two-point velocity correlation tensor of Eq. (2). CORR operates on a matrix U containing the fluid velocity components in all the grid points and time steps, and stores the results in a matrix R_{ij}. U is a matrix of size $NCOMP \times NMAX \times ITMAX$, where $NCOMP$ is the number of the velocity components considered ($NCOMP = 3$), $NMAX$ is the total number of grid points in which the velocity field has been evaluated ($NMAX = NX \times NY \times NZ$, the latter being the number of grid points in each of the the three directions x, y, z of D, respectively) and $ITMAX$ is the number of time steps of the temporal sequence of the velocity data. R_{ij} is a matrix of size $(NCOMP \times NMAX) \times (NCOMP \times NMAX)$.
- Phase TRAPEZ. Evaluation of the Fredholm integral of Eq. (2) on the computational domain D with the use of the trapezoidal rule, and conversion to the equivalent symmetric problem. TRAPEZ operates on the matrix R_{ij} and stores the results of the equivalent symmetric problem again in R_{ij}. It also calculates the weight functions of the trapezoidal rule (vector Ω).
- Phase TRED2. Execution of the tridiagonalization of matrix R_{ij} with the use of the Householder method. TRED2 operates on R_{ij} and stores the results in two vectors, d and e, that represent the diagonal and the sub-diagonal of the equivalent tridiagonal matrix, respectively. The algorithm used in this phase is the Householder algorithm [30].
- Phase TQLI (Tridiagonal QL Implicit). Solution of the eigenvalue problem (2) with the determination of eigenfunctions and eigenvalues. TQLI implements the QL algorithm with implicit shifts [30] to determine eigenvalues and eigenvectors of the real, symmetric and tridiagonal matrix calculated by TRED2. At the end of the process, d contains the eigenvalues and R_{ij} contains the eigenfunctions. The number of iterations for the evaluation of the first eigenvalues is 4–5, while much less effort is required for the calculation of the last ones. The average number of iterations required is 1.3–1.6.
- Phase TESTENERG. Execution of a test to verify the correct representation of the energy content by the eigenvalues [Eq. (6)]. TESTENERG only operates on matrix U and vector d.
- Phase COEFF. Evaluation of the time-dependent coefficients $a_n(t)$ of Eq. (3) by means of a standard inversion procedure. COEFF operates on matrices U and R_{ij}, and stores the results in matrix A of size $NCOMP \times NMAX \times ITMAX$.

Table I Asymptotic Complexity of Different Phases of the POD algorithm

Phase	Floating-point operations	Preliminary test (s)
CORR	$2 \times ITMAX \times (NCOMP \times NMAX)^2$	3.0556
TRAPEZ	$2 \times (NCOMP \times NMAX)^2$	0.1394
TRED2	$\frac{4}{3} \times (NCOMP \times NMAX)^3$	49.3152
TQLI	$3 \times (NCOMP \times NMAX)^3 + 30 \times (NCOMP \times NMAX)^2$ *(for eigenvectors)* *(for eigenvalues)*	20.6847
TESTENERG	$2 \times ITMAX \times NCOMP \times NMAX$	0.0515
COEFF	$ITMAX \times NCOMP \times NMAX^2\ (4NCOMP + 3)$	1.9830
Minor Operations		0.5318
		Total = 75.7633

The asymptotic complexity of different phases of the procedure is experimentally evaluated, by considering the number of floating-point operations that characterize each phase. The results are reported in the second column of Table I. Of all the phases shown in Table I, TRED2 and TQLI exhibit the highest degree of complexity. Thus, in the formulation of the parallelization strategy, data distribution, load balancing and communications have to be designed in such a way as to first optimize TRED2 and TQLI. The results of a preliminary test performed on a Pentium MMX 200 MHz in the case of $NX = 5$, $NY = 5$, $NZ = 5$ and $ITMAX = 10$, are reported in the third column of Table I (execution time in seconds).

More than 92% of the computing time is spent for the execution of TRED2 and TQLI. For problems of higher dimensions it can be inferred that the relative weight of these two phases will increase, due to the fact that the exponential terms of their asymptotic complexities are of higher order with respect to those of the other phases.

4. MEMORY ALLOCATION

The following expression of experimental nature, for the memory allocation (still sequential), is devised:

$$M_{seq} \approx NCOMP^2 \times NMAX^2 + 2 \times NCOMP \times ITMAX \times NMAX\ . \tag{7}$$

It appears that the memory requirements, rather than on the extent of the temporal sequence of data, depend almost exclusively on the size of the computational

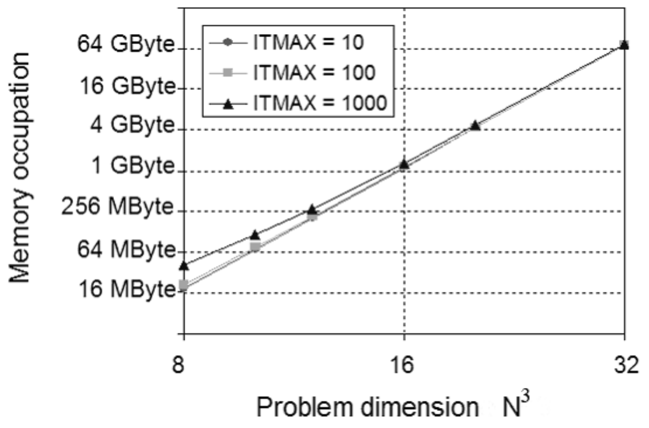


Fig. 1 Memory occupation of the sequential procedure with the dimensions of a cubic domain at different values of *ITMAX* (log-scales).

domain. Evaluations of preliminary nature (not reported) have shown that the first term on the right-hand side of (7) (due to the matrix of the correlation tensor) counts for 97% of the total memory allocation. The second term weights for the remaining 3% and is due to the matrices of the velocity data and the time dependent coefficients (minor terms have been neglected). In Fig. 1, the behavior of M_{seq} with problem dimension N^3 is shown at different values of *ITMAX*. The three curves practically coincide for $N^3 > 16$.

In developing the parallel code, the matrix of the correlation tensor has to be distributed among the n available processors. Thus, a portion of matrix equal to (R_{ij}/n) is attributed to each processor in such a way as the memory occupation of R_{ij} results independent on the number of processors. For what matrices U and A are concerned, it has been verified that it is not convenient to have a copy of them on each processor, due to a not-acceptable RAM overhead connected to this choice. Thus, matrices U and A are also distributed among the processors so that, in the whole, the memory allocation of the parallel code results not that different from the sequential version.

5. PARALLELIZATION STRATEGY

In devising a parallelization strategy, an optimal global parallelization is sought, i.e., minimization of the communications and optimal balancing of the computational load among the processors.

- Phase CORR. For the evaluation of the two-point correlation matrix R_{ij}, CORR utilizes the fluid velocity components stored in matrix U. Both ma-

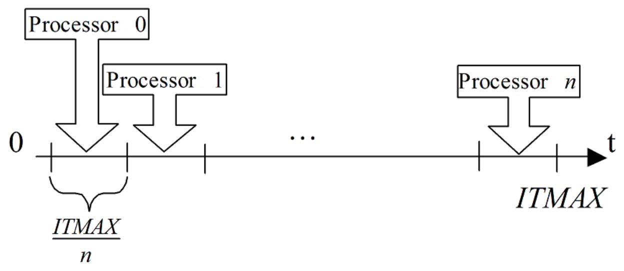


Fig. 2 Distribution among the n available processors of matrix U in ($ITMAX/n$) groups of instants, per processor.

trices R_{ij} and U have to be distributed among the available processors (Section 4), but the execution of CORR mainly depends on how matrix U — rather than R_{ij} — is distributed among the processors. To simplify the parallelization of CORR, matrix U is divided along the time axis. A subset of the fluid velocity components corresponding to a given time interval ($ITMAX/n$) is attributed to each processor, as shown in Fig. 2 (n is the number of processors). With this partition, in order for all the processors to continue in the evaluation of R_{ij}, each processor cyclically has to communicate to the other processors its portion of U.

- Phase TRAPEZ. In this phase, the Fredholm integral of (2) is calculated by using the trapezoidal rule and R_{ij} is transformed into the equivalent symmetric problem. The quadrature formula utilizes appropriate weight functions stored in the vector Ω (dimension $NMAX$) that occupies a limited amount of memory. The vector Ω is duplicated onto each processor, so that no communications among processors are needed in this phase. The evaluation of the integral results perfectly parallel.
- Phase TRED2. The calculations incorporated in this phase can be logically divided in two parts: *i*) tridiagonalization of the input matrix R_{ij} (the most relevant part) with memorization of the transformation vectors in R_{ij} (while it goes to zero in the course of the calculations) and *ii*) reconstruction of the rotation matrix Q for the transformation of the symmetric input matrix R_{ij} into the tridiagonal matrix $QR_{ij}Q^T$ (^{T} is the transpose).

For the parallelization of the first part of TRED2 one could divide the input matrix among the processors by rows or columns, due to the fact that the matrix is symmetric. In this way, with the advancement of the cycle, the sub-matrix onto which the calculations are executed becomes progressively smaller (Fig. 3). At the beginning the matrix does not contain empty rows or columns. At step k only the rows and columns with indices 1 to k contain data. This happens because, in going from step k to step k + 1, the row and column k go to zero (exception made for the

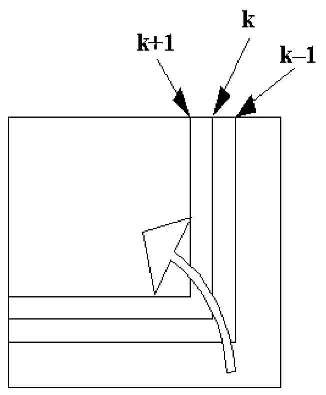


Fig. 3 Scheme of the reduction of matrix R_{ij} during the process of tridiagonalization (steps $k - 1$, k, $k + 1$).

elements of the diagonal and sub-diagonal that are memorized in vectors d and e). Thus, with a distribution of the input matrix in consecutive rows, the computational load among the processors results unbalanced. At step k the processors that have no rows (or columns) with index less than k are not involved in the calculations and the parallelism decreases while the process advances. In order to avoid this situation, a solution called *cyclic parallelization* is devised. The solution can be implemented through rows or columns of R_{ij}, because of symmetry. With reference to the rows of R_{ij}, they are distributed cyclically among the processors in the following way (Fig. 4). The first processor receives row 0, row n, row $2n$, row $3n$, ... The second processor receives row 1, row $(n + 1)$, row $(2n + 1)$, row $(3n + 1)$, ... The third processor receives row 2, row $(n + 2)$, row $(2n + 2)$, row $(3n + 2)$, ..., and so on. Each processor receives in total a number of rows (nr/n, the number of rows nr divided by the number of processors n). In this way, the load of each processor increases uniformly and the parallelism remains constant.

The tridiagonalization procedure iterates on each row and on the corresponding column of the input matrix R_{ij}, setting to zero the required elements. For the execution of this operation some working quantities are evaluated, the scalar H and the vector u, of dimension equal to the side (row or column) of matrix R_{ij}. These operations are executed by a single processor (each row belongs to a unique processor). The vector $p = R_{ij}u/H$ is then evaluated. Since u and H have been computed by a single processor, it is necessary to send both of them to the other processors, in order to enable the processors themselves to calculate their own portion of vector p. The next step is the calculation of the product $u^T p$. Being p distributed, each processor first executes only a part of the calculation. Then, the sum among the processors is performed, to calculate $K = u^T p/2H$. Now, each processor is in the condition

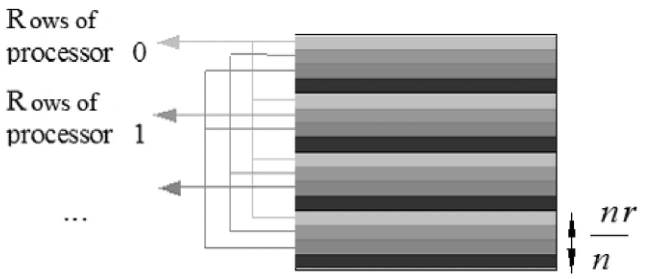


Fig. 4 Cyclic distribution among the processors of matrix R_{ij} by rows.

of evaluating its own portion of vector $q = p - Ku$. In order to perform the final calculation for the updating of matrix R_{ij}, all the processors must possess the entire vector q. Thus, each processor has to communicate to the others its portion of q. The final operation is given by:

$$R_{ij}^{i+1} = R_{ij}^{i} - qu^{T} - uq^{T}. \tag{8}$$

and is executed independently by the processors. During the steps outlined above, the elements of the diagonal and sub-diagonal that have been obtained are stored in vectors d and e, also cyclically distributed among the processors.

For what the second part of TRED2 is concerned, matrix Q is given by the accumulated transformation ($N = NCOMP \times NMAX$):

$$Q = P_1 \times P_2 \times ..... \times P_N, \tag{9}$$

where:

$$P_j = \left(1 - \frac{uu^T}{H}\right)_j. \tag{10}$$

To obtain Q, the following recursive operations are performed:

$$Q_N = P_N \tag{11}$$

$$Q_j = P_j \times Q_{j+1} \quad (j = N - 1. ......, 1) \tag{12}$$

$$Q = Q_1 \tag{13}$$

where the product (12) can be written as:

$$Q_j = \left(1 - \frac{uu^T}{H}\right)_j Q_{j+1} = Q_{j+1} - \left(\frac{uu^T}{H}\right)_j Q_{j+1} . \tag{14}$$

For the execution of this calculation, the vectors u and the scalars *H* obtained in each iteration have to be memorized within the cycle of tridiagonalization of the input matrix. The vectors *u* are memorized in the rows of the matrix that becomes progressively empty, owing to the ongoing process of reduction. The *Hs* are stored in a vector. In order for each processor to execute in parallel the evaluation of Q_j, all processors must receive, in each iteration, the vector *u* by means of an operation of broadcasting of the *u*-possessing processor.

- Phase TQLI. In this phase, the eigenvalue problem associated with the matrix that is tridiagonalized during TRED2, is solved. In each iteration, the calculations are logically divided in two steps: i) calculation of the i-th eigenvalue and *ii*) updating of the correspondent eigenvector.

The process of evaluation of the eigenvalues is characterized by a complexity of one order of magnitude less than the calculation of the eigenvectors (Table I). Thus, the calculation of the eigenvalues is duplicated onto the *n* available processors, while the updating of the eigenvectors is parallelized. For the parallelization of this step, the diagonal and the sub-diagonal of R_{ij} (vectors *d* and *e*) have to be communicated to all the processors. At the end of TRED2 the two vectors are distributed, so that each processor has to send to the others its portion of *d* and *e*. This operation is not particularly heavy, due to the limited dimensions of both *d* and *e* (*NCOMP* × *NMAX*). It can be noted that the result of zero communications among the processors can be reached in the parallelization of this part of the calculations if the matrix of the correlation tensor is distributed among the processors *by rows*. This is due to the fact that a dependency exists in this direction, in the sense that the element $(i, j + 1)$ depends on the element (i, j). It can also be noticed that: *i*) if the distribution of the matrix of the correlation tensor would be by columns, it would be necessary to communicate the columns at the boundaries; *ii*) if the distribution of the matrix of the correlation tensor would be *cyclic* by columns, it would be necessary to communicate the entire matrix.

- Phase TESTENERG. This phase is not particularly complex. The calculations are executed by all the processors in parallel, being *U* distributed among the processors along the temporal axis up to *ITMAX*, as outlined in the description of CORR.
- Phase COEFF. For this phase the processors have to be in the condition of independently operating on their own portion of R_{ij}. Thus, it is necessary that all the processors possess the portion of *U* corresponding to the given

time *i*. The processor possessing the portion of U at $IT = i$ must communicate it to all the others. The most internal cycle of the procedure goes from 1 to NCOMP. This cycle multiplies three elements of matrix R_{ij}, which belong to three consecutive rows. Thus, in order to realize an efficient parallelization, a *cyclic partition* of the matrix of the correlation tensor among the processors is again needed, this time in groups formed by *three consecutive rows*.

Overall, the most relevant factors for the parallelization of the whole POD procedure are the optimization of TRED2 and TQLI, with some additional requirements related to COEFF. For TRED2, the so-called *cyclic distribution* of the matrix R_{ij} among the processors is mainly needed. For TQLI it is needed that *an entire row* be avaliable to each processor. In the case of COEFF, at least *three consecutive rows* are needed for each processor. On this basis, the idea implemented in this work for the formulation of an optimal solution for the parallelization of the highly composite POD code, is the so-called *cyclic distribution of the matrix of the correlation tensor in groups of three consecutive rows* onto each of the n processors. Moreover, CORR requires the distribution of matrix U along the time axis.

6. COMPUTING SYSTEM

The computer used for the performance tests is an HP 9000-V2500 (Fig. 5). It includes 20 processors HP PA-RISC 8500 all arranged in a single hypernode, totaling 16 GByte of RAM and 180 GByte of mass memory (up to 16 processors have been used in the present work). The processor PA-RISC 8500 is a 64 bit processor with 440 MHz of frequency clock, 1 MByte of data cache on-chip and 0.5 MByte of in-

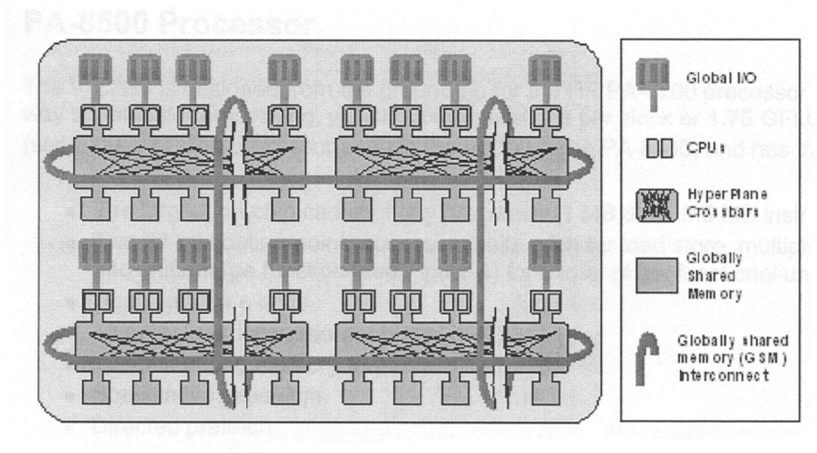


Fig. 5 Sketch of the computing architecture.

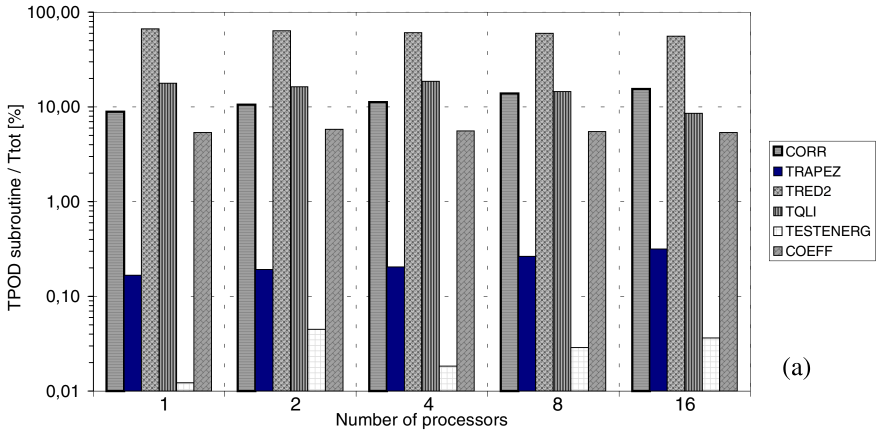
struction cache on-chip per processor and 2 clock maximum latency to cache. The memory architecture is a crossbar-based symmetric multiprocessor (SMP) of 8 × 8 non-blocking multiported crossbar type. The maximum total bandwidth allowed is 15.36 GByte/s. The processors are connected in quadruplets through a 768 MByte/s memory bus to the Processor Agent Controller (PAC). The crossbar (4 Routing Arrays Controllers, RACs) connects the PACs of the hypernode with all the controllers for memory access (MACs). The MACs control the access from the crossbar to all memory units, both local and remote with respect to a given hypernode. All memory transfers are managed by Toroidal Access Controllers (TACs) connected to the MACs. The hypernode is equipped with 8 (120 MByte/s) standard PCI (Peripheral Component Interface) buses for I/O, controlled by a PCI-bus Interface Controller (PIC) for each PAC. The operating system is the HP-UX (HP's industry-standard UNIX operating system).

7. RESULTS

The parallel performance of the POD code in the whole and of its different parts is investigated for two different datasets: *i*) dataset *A* with *NMAX* = 250 (*NX* = 10, *NY* = 5, *NZ* = 5) and *ITMAX* = 50 (*NCOMP* = 3); *ii*) dataset *B* with *NMAX* = 500 (*NX* = 10, *NY* = 10, *NZ* = 5) and *ITMAX* = 50 (*NCOMP* = 3). Groups of 2, 4, 8, and 16 processors have been used in the calculations, besides the sequential version of the code.

In Figs 6a,b, the temporal allocation (% of the total execution time) of the six phases of the code described in Section 5 (for both datasets) is shown with the number of processors. The behavior is similar for the two datasets. As an example, in the case of dataset *B* it can be noticed that four phases significantly influence the execution time, TRED2, TQLI, CORR, and COEFF. The remaining two phases weigh for less than 1% of the total execution time. TRED2 uses 68% of the total time on average, slightly decreasing with the number of processors (from 72% with one processor to 64% with 16 processors). TQLI uses 15% of the total time on average, also decreasing with the number of processors (from 19% with one processor to 13% with 16 processors). CORR, on the contrary, increases its execution time with the number of processors, going from 4% with one processor to 12% with 16 processors. COEFF uses 3% of the total time, practically constant with the number of processors.

In Figs. 7a,b and 8a,b, speed-ups and efficiencies of the four more relevant phases of the calculations separately considered (TRED2, TQLI, CORR, COEFF) with the number of processors are shown for both datasets. The speed-up is defined as the run-time with one processor divided by the run-time with a given number of processors. The efficiency is defined as the ratio between the actual and the ideal speed-ups.



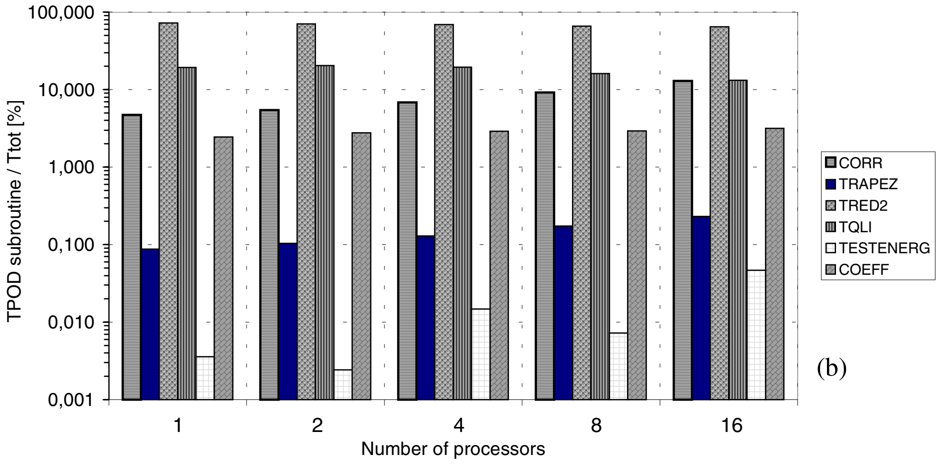
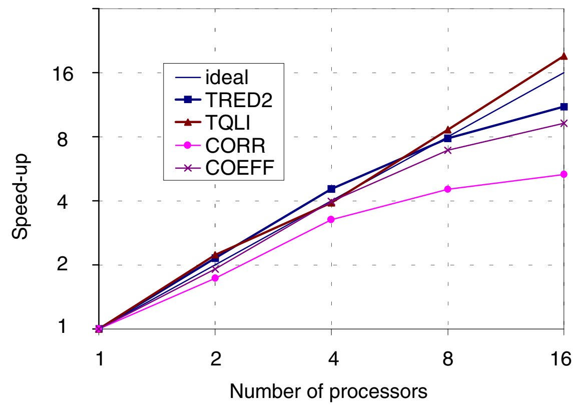


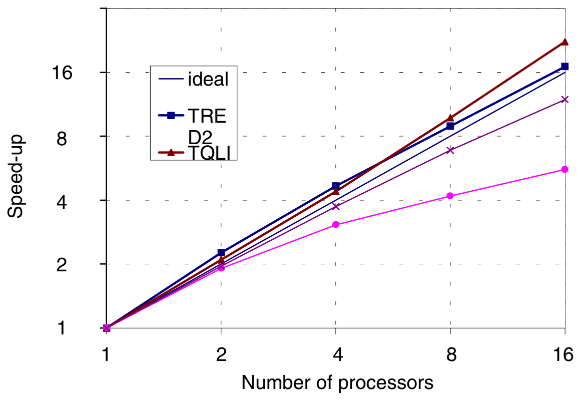
Fig. 6 Temporal allocation (% of the execution time) of the six phases of the POD procedure with the number of processors (log-scales): a) dataset *A*; b) dataset *B*.

Figure 7a shows the speed-up of the four most relevant phases of the procedure with the number of processors for dataset *A*. TQLI exhibits a superlinear behavior. TRED2 is superlinear up to 4 processors, is linear with 8 processors and decays with 16 processors. COEFF and CORR remain under the linear limit up to 16 processors.

Figure 7b shows the speed-up of the four relevant phases of the procedure with the number of processors for dataset *B*. In this case both TQLI and TRED2 exhibit a superlinear (or almost linear) behavior, up to 16 processors. The superlinear



(a)

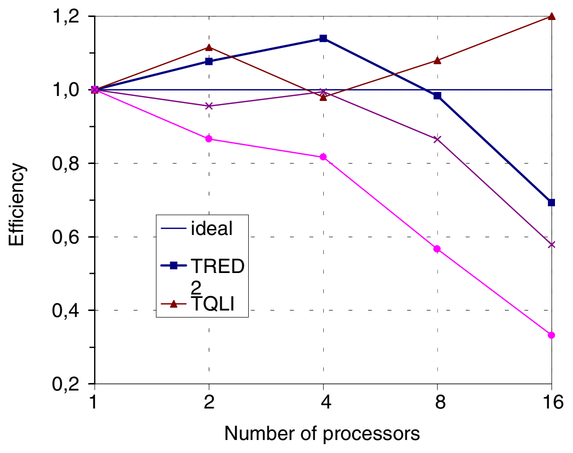


(b)

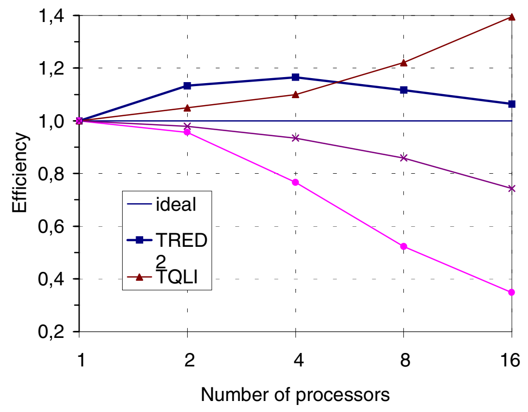
Fig. 7 Speed-up of the four most relevant phases of the POD procedure separately considered with the number of processors (log-scales): a) dataset *A*; b) dataset *B*.

results are related to the involvement of the cache. By increasing the amount of the processors, the memory required by each processor decreases and so do the cache-miss events. The practically linear behavior of TQLI is a permanent factor in the execution of the program, due to the absence of significant communications during the TQLI phase. Thus, a similar behavior is expected also in the calculation of problems larger than those tested.

In Figs. 8a,b, the aforementioned results are mirrored in terms of efficiencies instead of speed-ups. Figures 8a and b show the efficiency of the four relevant phases of the procedure separately considered, with the number of processors for both datasets. The superlinear behavior of TQLI is evident and also the fact that the efficiency of TRED2 increases in the case of dataset *B* with respect to dataset *A*. The efficiency of COEFF increases in the case of dataset *B* with respect to dataset



(a)

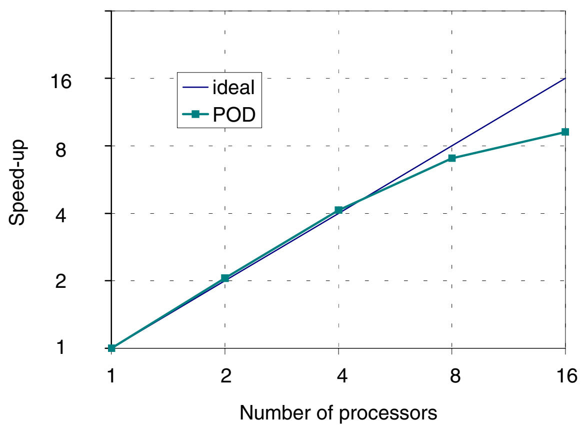


(b)

Fig. 8 Efficiency of the four most relevant phases of the POD procedure separately considered with the number of processors (semilog-scales): a) dataset *A*; b) dataset *B*.

A (about 75% with 16 processors for dataset *B*). The efficiency of CORR is almost the same (relatively low) for both datasets (about 35% with 16 processors), slightly increasing in the case of dataset *B* with respect to dataset *A*. Additional tests (not reported) have shown that the efficiency of CORR actually increases with the number of grid points of the computing domain, an encouraging result in the perspective of handling larger computational problems.

In Figs, 9a,b and 10a,b, the speed-ups and the efficiencies of the whole POD code with the number of processors, are reported for both datasets. Figures 9a and b



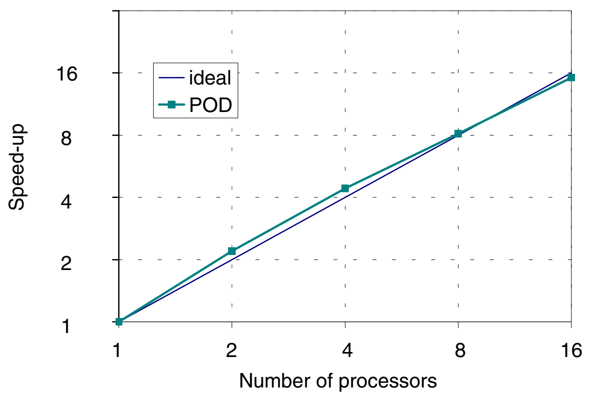
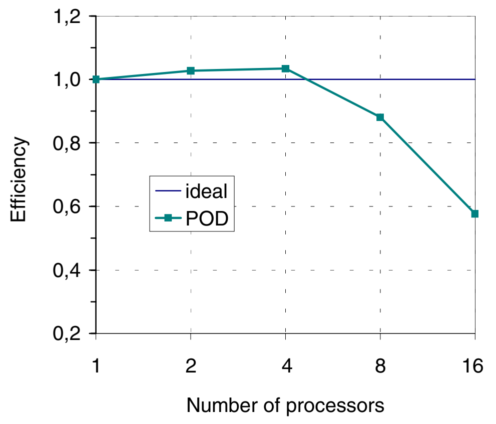


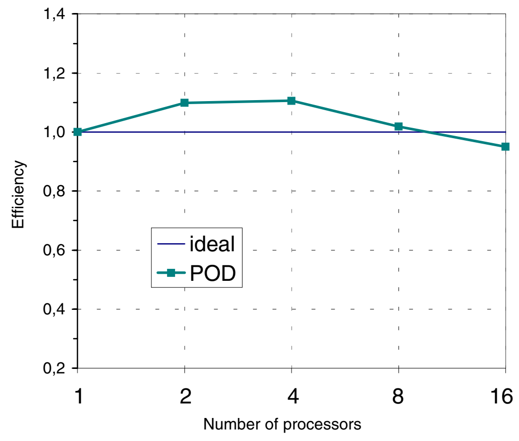
Fig. 9 Speed-up of the whole POD code with the number of processors (log-scales): a) dataset *A*; b) dataset *B*.

report the speed-ups, while in Figs. 10a and b the corresponding efficiencies are shown. In the case of dataset *A*, the speed-up of the whole code is nearly linear up to 4 processors and then the efficiency reaches the (relatively low) level of about 60% with 16 processors. In the case of dataset *B*, the speed-up is nearly linear up to 8 processors and then the efficiency reaches the (almost ideal) level of about 95% with 16 processors.

Overall, the superlinear character of phases TRED2 and TQLI compensate the behavior of subroutines COEFF and CORR, finally resulting in satisfactory levels of speed-up and efficiency for the whole POD code.



(a)



(b)

Fig. 10 Efficiency of the whole POD code with the number of processors (semilog-scales): a) dataset *A*; b) dataset *B*.

8. CONCLUDING REMARKS

A parallel computational code for the execution of the Proper Orthogonal Decomposition of turbulent flows in fluid dynamics is developed and its parallel performance tested. The POD procedure is an important tool in turbulent research, in the sense that it allows the selection of the most energetic modes of a turbulent flow. The POD code is highly composite, in the sense that it involves several computational phases of different mathematical nature. Computational tests are performed on a HP 9000-V2500 computer, involving up to 16 processors and related to two different datasets. Overall, the code exhibits satisfactory levels of speed-up and efficiency with an increasing number of processors involved in the calculations. This result is mainly due to the superlinear behavior of some of the phases of the calculations which compensate for the performace of other parts of the code.

REFERENCES

1. G. Alfonsi, G. Passoni, L. Pancaldo, and D. Zampaglione, A Spectral-Finite Difference Solution of the Navier–Stokes Equations in Three Dimensions, *Int. J. Num. Meth. Fluids*, vol. 28, pp. 129–142, 1998.
2. G. Passoni, G. Alfonsi, and M. Galbiati, Analysis of Hybrid Algorithms for the Navier–Stokes Equations with Respect to Hydrodynamic Stability Theory, *Int. J. Num. Meth. Fluids*, vol. 38, pp. 1069–1089, 2002.
3. C. G. Speziale, Analytical Methods for the Development of Reynolds-Stress Closures in Turbulence, *Ann. Rev. Fluid Mech.*, vol. 23, pp. 107–157, 1991.
4. M. Lesieur and O. Métais, New Trends in Large-Eddy Simulation of Turbulence, *Ann. Rev. Fluid Mech.*, vol. 28, pp. 45–82, 1996.
5. P. Moin and K. Mahesh, Direct Numerical Simulation: A Tool in Turbulence Research, *Ann. Rev. Fluid Mech.*, vol. 30, pp. 539–578, 1998.
6. G. Passoni, G. Alfonsi, G. Tula, and U. Cardu, A Wavenumber Parallel Computational Code for the Numerical Integration of the Navier–Stokes Equations, *Parall. Comput.*, vol. 25, pp. 593–611, 1999.
7. G. Passoni, P. Cremonesi, and G. Alfonsi, Analysis and Implementation of a Parallelization Strategy on a Navier–Stokes Solver for Shear Flow Simulations, *Parall. Comput.*, vol. 27, pp. 1665–1685, 2001.
8. P. F. Fischer and A. T. Patera, Parallel Simulation of Viscous Incompressible Flows, *Ann. Rev. Fluid Mech.*, vol. 26, pp. 483–527, 1994.
9. S. K. Robinson, Coherent Motions in the Turbulent Boundary Layer, *Ann. Rev. Fluid Mech.*, vol. 23, pp. 601–639, 1991.
10. R. L. Panton, Overview of the Self-Sustaining Mechanisms of Wall Turbulence, *Prog. Aerosp. Sci.*, vol. 37, pp. 341–383, 2001.
11. G. Alfonsi, Coherent Structures of Turbulence: Methods of Eduction and Results, *Appl. Mech. Rev.*, vol. 59, p. 307, 2006.
12. B. Lang, Efficient Eigenvalue and Singular Value Computations on Shared Memory Machines, *Parall. Comput.*, vol. 25, pp. 845–860, 1999.
13. R. Suda, A. Nishida, and Y. Oyanagi, A High-Performance Parallelization Scheme for the Hessenberg Double Shift QR Algorithm, *Parall. Comput.*, vol. 25, pp. 729–744, 1999.
14. J. S. Reeve and M. Heath, An Efficient Parallel Version of the Householder-QL Matrix Diagonalization Algorithm, *Parall. Comput.*, vol. 25, pp. 311–319, 1999.

15. J. L. Lumley, *Stochastic Tools in Turbulence*, Academic Press, New York, 1971.
16. L. Sirovich, Turbulence and the Dynamics of Coherent Structures. Parts I–III, Quart. *Appl. Math.*, vol. 45, pp. 561–590, 1987.
17. G. Berkooz, P. Holmes, and J. L. Lumley, The Proper Orthogonal Decomposition in the Analysis of Turbulent Flows, *Ann. Rev. Fluid Mech.*, vol. 25, pp. 539–575, 1993.
18. L. Sirovich and H. Park, Turbulent Thermal Convection in a Finite Domain. Part I. Theory, *Phys. Fluids*, vol. A2, pp. 1649–1658, 1990.
19. H. Park and L. Sirovich, Turbulent Thermal Convection in a Finite Domain. Part II. Numerical Results, *Phys. Fluids*, vol. A2, pp. 1659–1668, 1990.
20. L. Sirovich and A. E. Deane, A Computational Study of Rayleigh–Bénard Convection. Part II. Dimensions Considerations, *J. Fluid Mech.*, vol. 222, pp. 251–265, 1991.
21. L. Sirovich, M. Kirby, and M. Winter, An Eigenfunction Approach to Large-Scale Transitional Structures in Jet Flows, *Phys. Fluids*, vol. A 2, pp. 127–136, 1990.
22. M. Kirby, J. Boris, and L. Sirovich, An Eigenfunction Analysis for Axisymmetric Jet Flow, *J. Comput. Phys.*, vol. 90, pp. 98–122, 1990.
23. R. E. A. Arndt, D. F. Long, and M. N. Glauser, The Proper Orthogonal Decomposition of Pressure Fluctuations Surrounding a Turbulent Jet, *J. Fluid Mech.*, vol. 340, pp. 1–33, 1997.
24. S. V. Gordeyev and F. O. Thomas, Coherent Structure in Turbulent Planar Jet. Part I. Extraction of Proper Orthogonal Decomposition Eigenmodes and Their Self-Similarity, *J. Fluid Mech.*, vol. 414, pp. 145–194, 2000.
25. N. Aubry, P. Holmes, J. L. Lumley, and E. Stone, The Dynamics of Coherent Structures in the Wall Region of a Turbulent Boundary Layer, *J. Fluid Mech.*, vol. 192, pp. 115–173, 1988.
26. P. Moin and R. D. Moser, Characteristic-Eddy Decomposition of Turbulence in a Channel, *J. Fluid Mech.*, vol. 200, pp. 471–509, 1989.
27. L. Sirovich, K. S. Ball, and L. R. Keefe, Plane Waves and Structures in Turbulent Channel Flow, *Phys. Fluids*, vol. A2, pp. 2217–2226, 1990.
28. K. S. Ball, L. Sirovich, and L. R. Keefe, Dynamical Eigenfunction Decomposition of Turbulent Channel Flow, *Int. J. Num. Meth. Fluids*, vol. 12, pp. 585–604, 1991.
29. G. A. Webber, R. A. Handler, and L. Sirovich, The Karhunen–Loeve Decomposition of Minimal Channel Flow, *Phys. Fluids*, vol. 9, pp. 1054–1066, 1997.
30. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in Fortran 77*, Cambridge University Press, Cambridge, 1992.