Noname manuscript No.
(will be inserted by the editor)

Parallel QR algorithm for large-scale data-driven flow-field decompositions

with application to transitional boundary layers

Taraneh Sayadi · Peter J. Schmid

Received: date / Accepted: date

Abstract Many fluid flows of engineering interest, though very complex in appearance, can be approximated by low-order models governed by a few modes, able to capture the dominant behavior (dynamics) of the system. This feature has fueled the development of various methodologies aimed at extracting dominant coherent structures from the flow. Some of the more general techniques are based on data-driven decompositions, most of which rely on performing a singular value decomposition (SVD) on a formulated snapshot (data) matrix. The amount of experimentally or numerically generated data expands as more detailed experimental measurements and increased computational resources become readily available. Consequently, the data-matrix to be processed will consist of far more rows than columns, resulting in a so-called tall-and-skinny (TS) matrix. Ultimately, the SVD of such a TS data-matrix can no longer be performed on a single processor and parallel algorithms are necessary. The present study employs the parallel TSQR algorithm of [1], which is further used as a basis of the underlying parallel SVD. This algorithm is shown to scale well on machines with a large number of processors and, therefore, allows the decomposition of very large data-sets. In addition, the simplicity of its implementation and the minimum required communication makes it suitable for integration in existing numerical solvers and data-decomposition techniques. Examples that demonstrate the capabilities of highly parallel data-decomposition algorithms include transitional processes in compressible boundary layers without and with induced flow separation.

Keywords Parallel algorithms · data decomposition · computational fluid dynamics

Taraneh Sayadi
Department of Aerospace Engineering, University of Illinois at Urbana-Champaign, Urbana 61801, IL
Tel.: +1217-300-0901
E-mail: sayadi@illinois.edu
Department of Mathematics, Imperial College London, London SW7 2AZ, UK

Peter J. Schmid
Department of Mathematics, Imperial College London, London SW7 2AZ, UK

1 Introduction

In order to develop a correct and encompassing understanding of fluid flows, a detailed analysis of the observed flow phenomena is required. Ever growing high-performance computing (HPC) capabilities and high-quality experimental equipment have made studying complex flows possible, but at the same time, the sheer quantity of data that numerical and experimental efforts produce, puts considerable strain on common algorithms used to extract and analyze the principal flow structures and processes. This is particularly the case for data-driven decomposition techniques that involve the reduction and restructuring of the full data-matrix into dominant modes by linear algebra techniques. This methodology has recently been used in investigations of complex flows to either provide an explanation of the essential behavior of the system, or to generate reduced-order models based on a few dominant modes that optimally capture the full dynamics [2], [3], [4], [5], [6].

Data-driven decomposition techniques rely solely on the underlying data-set, thus rendering them applicable to numerical or experimental results alike and making them useful and versatile for post-processing purposes. Proper orthogonal decomposition (POD), for example, belongs to this class of methods, where structures are ranked by their L_2-content (energy, variance, etc.) and form an orthogonal basis for the analyzed data-set. Often, more than one frequency is attributed to each POD mode, since the optimization process is focused on the spatial content of the data rather than on the time evolution. Balanced POD [2,7] is an alternative to POD for input-output systems and constitutes a tractable method for computing approximate balanced truncations of large-scale state-space systems used in flow control applications. Another example of a data-decomposition technique is Koopman [8], [9] or dynamic mode [10] decomposition. The result of this decomposition algorithm is a non-orthogonal basis, where each extracted mode is associated with a specific frequency present throughout the data. DMD has recently been applied to various flow configurations (see, for example, [9], [3], [11]), extracting dominant frequencies from a given data-set. These data-driven decomposition techniques have also been used to perform parametric studies on dynamical systems going through bifurcation [12].

All of the above data-decomposition techniques contain a QR decomposition of the data-matrix. Besides being a critical component of various algorithms, the QR-factorization has a wide range of applications in many disciplines, from data-mining to efficient storage and retrieval of high-dimensional data [13], from costumer recommender systems [14] to multiple-input and multiple-output (MIMO) systems such as transmitters and receivers in radio transmissions [15]. In our case of fluid flow analysis, the size of the gathered data (either from large-scale simulations or highly resolved experimental measurements) or the synchronous processing of composite or parameter-dependent data results in snapshot data-matrices with an excessive number of rows (equal to the number of spatial or composite degrees of freedom) but only a rather small number of columns (equal to the number of snapshots). This type of matrices is referred to as tall-and-skinny (TS). The TS data-matrix will eventually render the QR-based data-driven algorithms computationally expensive and ultimately prohibitive to execute on a single processor. For this reason, it is necessary to design and implement a parallel routine for performing the decomposition algorithm; this routine would eliminate the solution of

a QR-decomposition of the full, large-scale data-set and consequently allow the decomposition algorithm to remain computationally tractable.

Even though other implementations of the parallel QR algorithm for the purpose of high-performance computing exist, for example, within the Trilinos [16] suite of computational libraries, we propose an alternative parallel algorithm as a stand-alone option that, due to its simplicity, can be more easily integrated into the underlying flow solver and used for post-processing applications. The ultimate goal of our undertaking is to propose an algorithm that shows good scaling over a large number of processors and has a minimal dependence on external libraries, making it easily adaptable to the underlying computing infrastructure.

We apply the proposed parallel algorithm to perform the QR-factorization, which will be used as a basis for the singular value decomposition (SVD) which, in turn, is needed for the data-driven POD and DMD decompositions. The algorithm will be demonstrated and tested on data from direct numerical simulations of transitional and turbulent boundary layers [17], with and without pressure-induced separation, details of which are presented in Section 3.

2 Parallel algorithm

In this section we first introduce the parallel QR-decomposition algorithm that is further used as a building block for the SVD-based data-driven decomposition techniques. The algorithm for finding the singular values and singular vectors is also described. The scaling of the developed algorithm is tested on up to 1024 processors. For the purpose of this study, the parallel SVD algorithm is used as the initial step of performing DMD [10], hence, additional steps of parallelizing the DMD-based decomposition is outlined. However, the developed SVD algorithm can also be used to perform other SVD-based data-driven decompositions, such as POD.

2.1 Parallel QR factorization

The QR-factorization and SVD are two fundamental decompositions that have many applications in scientific computing and data analysis. The TSQR algorithm of [1] represents the cornerstone of our parallel DMD algorithm; it was initially applied in the context of DMD using the MapReduce programming framework by [18]. A schematic of the algorithm is shown in Figure 1, and the algorithm is presented in the appendix (algorithm 1). The algorithm first divides the data-matrix into sub-matrices of the same column size and distributes them among the processors. There are no restrictions on how to assemble and divide the data-matrix, and a non-uniform breakup of the full data-matrix is conceivable. On each processor, a QR-factorization is then performed on the respective sub-matrices and the resulting upper-triangular matrices $\mathbf{R}_i$ are gathered into a single matrix, $\mathbf{R}'$. This gathering constitutes the only required communication step between all processors in the full algorithm. An additional QR-factorization is then performed by each processor on the resulting $\mathbf{R}'$ matrix, which results in $\mathbf{Q}_2$ and $\mathbf{R}$. The matrix $\mathbf{R}$ is the final upper-triangular matrix of the desired QR-factorization of the full data-matrix. Each processor then retains its respective portion of $\mathbf{Q}_2$, $\mathbf{Q}_{2i}$

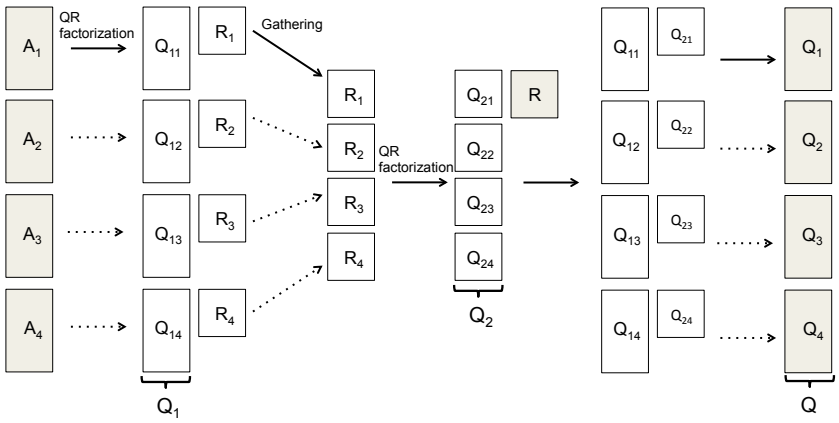


Fig. 1: Direct TSQR algorithm suggested by [18]. For simplicity four processors are illustrated in the graph. The algorithm simply carries over, as demonstrated, to larger number of processors.

with $i = 1, \cdots, m$, where m is the number of processors. $\mathbf{Q}_{2i}$ is a square matrix with the size of the number of snapshots. Multiplying the resulting $\mathbf{Q}_{2i}$ by the original $\mathbf{Q}_{1i}$'s of the sub-matrices yields the final $\mathbf{Q}$ of the full data-matrix, which is already distributed among the processors.

2.2 Parallel singular value decomposition

Once the QR-factorization is performed, the singular value decomposition of the data-matrix can be calculated in a straightforward manner by applying an additional SVD on the small matrix, $\mathbf{R}$, resulting in $(\mathbf{U}_\mathbf{R}, \mathbf{\Sigma}, \mathbf{W}^t) = \mathrm{SVD}(\mathbf{R})$, which is performed by all the processors. The left-singular vectors $\mathbf{U}_i$ of the full data-matrix are then computed by multiplying $\mathbf{U}_\mathbf{R}$ by the respective portion of the orthogonal matrix $\mathbf{Q}_i$, already available on each processor, $\mathbf{U}_i = \mathbf{Q}_i \mathbf{U}_\mathbf{R}$. The singular values and the right-singular vectors are already stored in $\mathbf{\Sigma}$, and $\mathbf{W}^t$, respectively. The described algorithm is also given in the appendix (algorithm 2).

In order to investigate whether the results of the parallel algorithm match those of the serial one, a singular-value decomposition is performed on a data-matrix consisting of 101 snapshots taken from a subdomain of a direct numerical simulation (DNS) of a transitional boundary layer, studied by [17], with 128, 100, and 128 grid points in the streamwise, wall-normal and spanwise directions, respectively. Figure 2 compares the singular values obtained from the parallel and serial algorithms; the results agree to within numerical round-off.

Figure 3 shows the scaling of the parallel algorithm as the number of processors increases. This scaling also includes the preceding QR factorization. Each column of the snapshot matrix consists of approximately eight million entries. This is the maximum size of data which would fit on 64 processors (lower limit of the scaling analysis). The maximum number of processors used for this scaling analysis is 1024. The figure shows that the scaling is near perfect for this case study.

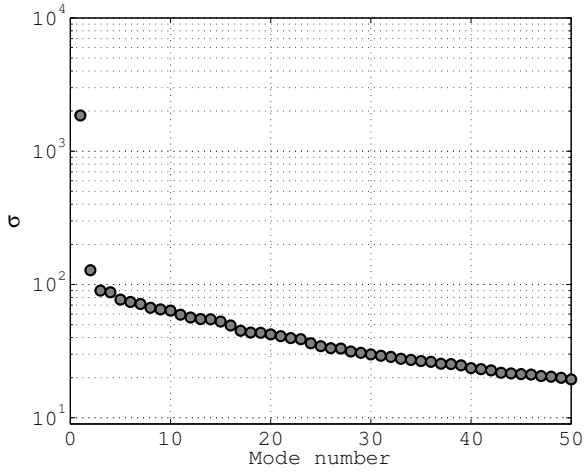


Fig. 2: Comparison of the singular values of the serial to the parallel SVD algorithm. ○, serial algorithm; •(gray), parallel algorithm.

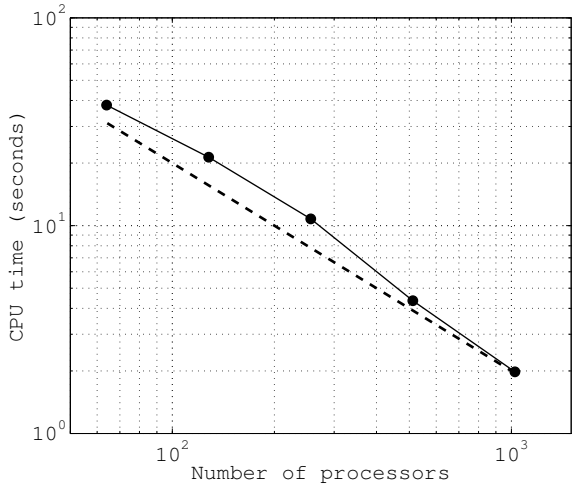


Fig. 3: The speed-up of the parallel algorithm. – – –, linear scaling; –•– scaling of the parallel algorithm.

To further analyze the scaling properties, the time spent on each portion of the parallel decomposition algorithm — from QR-factorization to the final SVD and DMD — is compared and reported in Table 1; we use a range of processors, from a single unit to up to 64 units. Each snapshot consists of approximately thirty million degrees of freedom, and data-matrix consist of eighteen snapshots. This table shows that as the number of processors increases the required time for performing the decomposition diminishes. There is about a factor of four speed-up

Number of processors	QR (s)	SVD (s)	Dynamic eigenvalues and eigenvectors (s)
1	40	20	30
16	10	4	8
32	5	3	4
64	2	1	3

Table 1: Comparison of the performance of the parallel QR-based algorithms to the serial version.

when 16 processors are used compared to only one. When the number of processors are increased further, each time by a factor of two, the required processing time reduces by the same factor, respectively.

2.3 Dynamic mode decomposition

Once singular values and vectors of the full data-matrix are computed, the same procedure as described in [10] can be followed. The data sequence of snapshots is given by the matrix $\mathbf{V}_1^N$ (the snapshot matrix, $\mathbf{V}_1^{N-1}$ is the same as $\mathbf{A}$, given in previous sections):

$$\mathbf{V}_1^N = \{v_1, v_2, \cdots, v_N\}, \tag{1}$$

where the column vectors v_i denote the i^{th} snapshot of the flow field containing, e.g., the velocity field. If we assume that a linear mapping $\mathbf{M}$ connects snapshot v_i to the subsequent snapshot v_{i+1}, we have

$$\mathbf{V}_2^N = \mathbf{M}\mathbf{V}_1^{N-1}. \tag{2}$$

Replacing the snapshot matrix by the singular values and vectors from the parallel SVD and multiplying both sides by $\mathbf{U}^*$ produces $\mathbf{U}^*\mathbf{M}\mathbf{U} = \mathbf{U}^*\mathbf{V_2^N}\mathbf{W}\mathbf{\Sigma^{-1}} \equiv \tilde{\mathbf{S}}$. In order to compute $\tilde{\mathbf{S}}$, $\mathbf{U}^*\mathbf{V}_2^N$ is initially formed on each processor separately and the resulting small matrices are then added across all processors. Let y_i and μ_i be the i^{th} eigenvectors and eigenvalues of $\tilde{\mathbf{S}}$, respectively. We then have

$$\mathbf{U}^*\mathbf{M}\mathbf{U}\mathbf{Y} = \tilde{\mathbf{S}}\mathbf{Y} = \mathbf{Y}\boldsymbol{\mu}, \tag{3}$$

and

$$\mathbf{M}\mathbf{U}\mathbf{Y} = \mathbf{U}\mathbf{Y}\boldsymbol{\mu}, \tag{4}$$

with $\mathbf{Y}$ containing (as columns) the eigenvectors y_i and $\boldsymbol{\mu}$ containing (on the diagonal) the eigenvalues μ_i. The dynamic modes are then calculated as

$$\mathbf{\Phi} = \mathbf{U}\mathbf{Y}. \tag{5}$$

The final multiplication (Eq. 5) can easily be performed in parallel, by multiplying each portion of $\mathbf{U}$, $\mathbf{U}_i$ by the matrix $\mathbf{Y}$. It can be deduced from Eqs. (2)-(5) that, once the singular values are computed, the algorithm is parallelizable in a straightforward fashion. Figure 4 shows a comparison of the eigenvalues of $\tilde{\mathbf{S}}$, evaluated via the serial and parallel algorithms. The subdomain is identical to the one used in Section 2.2 to compute the singular values in Figure 2. The algorithm describing steps (3 - 5) is given in the appendix (algorithm 3).

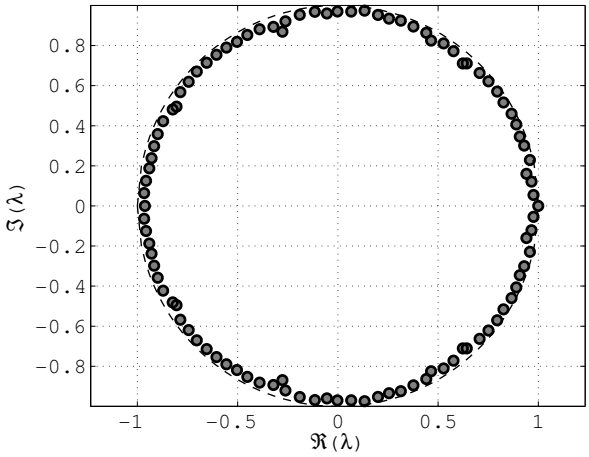


Fig. 4: Comparison of the eigenvalues of $\tilde{\mathbf{S}}$ using the serial and the parallel DMD algorithm. ○, serial algorithm; •(gray), parallel algorithm.

	N_x	N_y	N_z	No. (snapshots)	No. (processors)
separated boundary layer	600	200	128	291	120
classical boundary layer	4096	240	512	101	1024

Table 2: Computational domain of the respective direct numerical simulations used for the decomposition; total number of degrees of freedom per flow variable, per snapshot for these boundary layer simulations with and without separation is 15 million and 0.5 Billion grid points, respectively.

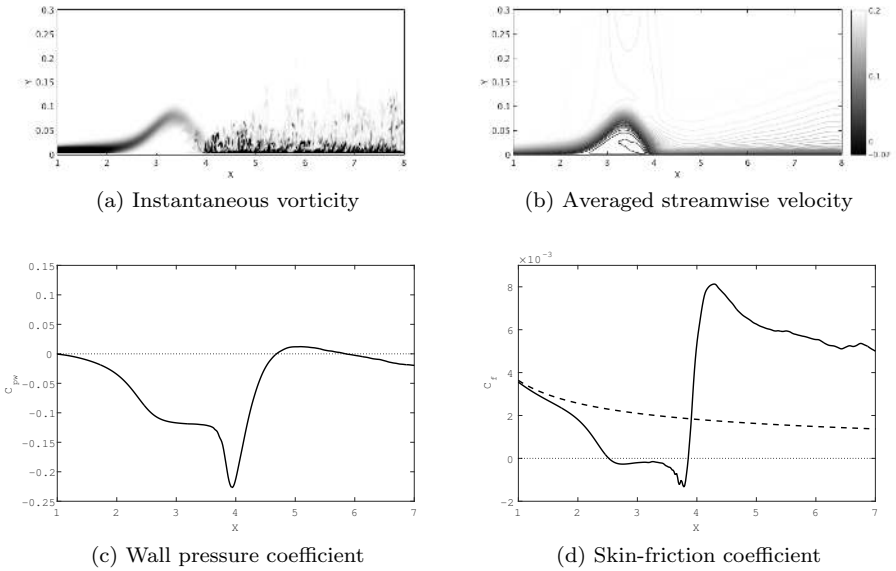
3 Application to large-scale data-sets

In this section we apply the described algorithm to data from direct numerical simulations of transitional and turbulent boundary layers; the first case (section 3.1) addresses the transition to turbulent fluid motion of a compressible, low-Mach number boundary layer over a flat plate in the presence of a separation bubble, while the second case (section 3.2) treats the classical transition process of a similar flow regime over a flat plate.

The details of the computational domains and resolutions used for each data-set are shown in Table 2. The extracted dynamics are then analyzed using the DMD methodology. We should note that the size of the two data-sets renders them too large to be decomposed without the parallelized algorithm introduced above.

3.1 Transition to turbulence in a compressible separated boundary layer

The separated boundary layer, presented in this study, was first analyzed by [19] using direct numerical simulation. The adverse pressure gradient, required to in-



(a) Instantaneous vorticity

(b) Averaged streamwise velocity

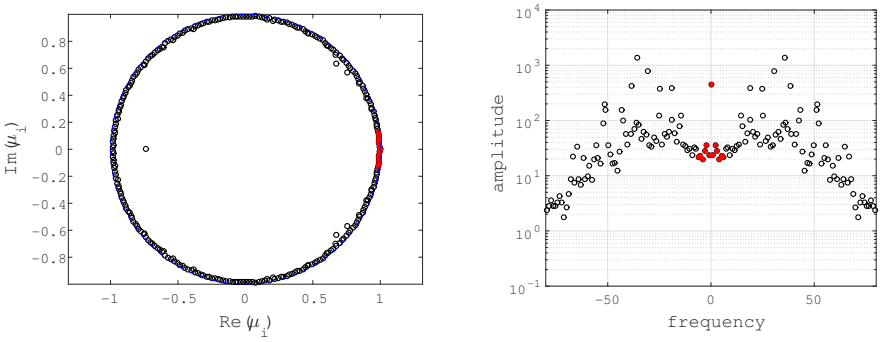
(c) Wall pressure coefficient

(d) Skin-friction coefficient

Fig. 5: Details of the separated boundary layer. (d) Skin-friction coefficient; − − −, laminar profile; ——, skin-friction profile of the separated boundary layer.

duce separation, is produced by means of suction through the wall-normal velocity at the top boundary, as proposed by [19]. This configuration was further employed by [20] to assess the performance of large eddy simulation in predicting separation. The computational domain and the governing equations used to conduct the direct numerical simulation here are similar to that described in [20]. As demonstrated in Figure 5(a) the presence of the separation bubble causes the boundary layer to transition. In addition, the boundary layer following the reattachment is thicker than the laminar boundary layer preceding the separation, as shown in Figure 5(b). These observations agree well with the simulations of [19] and [20]. The length of the separation bubble, estimated using Figure 5(d), $\Delta X_{\mathrm{SB}} = 1.4$, is shorter than the length predicted by [19]. The difference is due to a different suction profile at the upper boundary. The solver employed in the current study is compressible, as a result, the suction profile described in the incompressible framework of [19] is augmented by blowing with similar shape and an opposite sign to keep the total mass flux constant, and to allow the pressure gradient to relax back to zero away from the blowing and suction region, as demonstrated by Figure 5(c). Due to the added blowing, the separation bubble reattaches sooner than in the incompressible simulation, where only suction is applied.

In order to analyze the dynamics of the laminar separation, DMD is applied on two hundred and ninety equally spaced snapshots of the separating bubble. The resulting eigenvalues and spectrum are shown in Figure 6, where six modes are highlighted excluding the mean. These modes are selected using the sparsity promoting algorithm of [21]. The full decomposition is performed using 120 processors in 149 seconds.

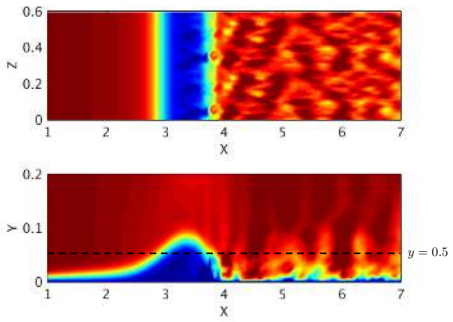


(a) Eigenvalues, μ_i (b) Spectrum

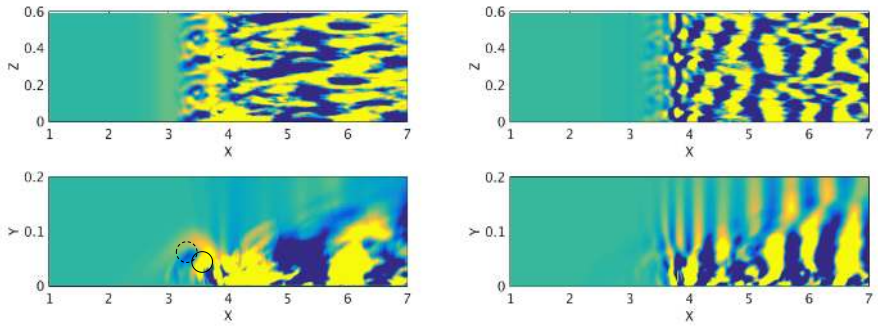
Fig. 6: Eigenvalues and the spectra of the dynamic modes of the separation bubble. Modes selected by the sparsity promoting algorithm are highlighted in red.

The combination of the six selected modes encompasses the low-frequency dynamics of this separated flow, and can be further explained from the shape of the modes and their effect on the mean profile, shown in Figure 7. Two slices in $x-z$ plane, at $y = 0.5$ on the top, and $x-y$ plane, at $z = 0.3$ on the bottom, are plotted. The combination of the two lowest frequency modes (2 & 3) are shown in Figure 7(b). These modes are active on the shear layer with a wavelength highlighted by the two circles in the bottom half of Figure 7(b). In addition, the $x-z$ slice (top half of the same figure) shows a pair of high and low velocity streaks, in the region of $3 \leq x \leq 4$, which resemble the legs of Λ-shaped vortices. These Λ-shaped structures are present in many transitional flows, one of which is described in the following section (K-type transition), and are characteristic of the appearance of the secondary instabilities preceding turbulence. Two of these structures are present in the spanwise length of the domain. These modes evolve into elongated streak-like structures following the reattachment point. These two modes contain the dynamics of shear layer oscillations including the presence of Λ-shaped vortical structures, which evolve into streaky structures downstream. The shape of the remaining modes is shown in Figure 7(c). In contrast to modes 2 & 3, these low-frequency modes are less pronounced at the edge of the shear layer and become more active near the reattachment point, defining the dynamics of the reattachment region. Moreover, these modes evolve into waves with certain wavelength in the streamwise direction, illustrated in the top half of Figure 7(c), rather than streaky structures. These wave-lengths manifest themselves in the form of rollers appearing downstream of the separation bubble, visible in the bottom half of Figure 7(a).

The modulation of the mean flow through these selected modes is plotted in Figure 7(a). This figure shows that the combination of these modes constitutes the flapping of the shear layer and the modulation of the reattachment line. Therefore, a few DMD modes, extracted using the parallel decomposition algorithm, are able to reconstruct the overall dynamics of the transitional region of the separated flow. This example is used as a proof of concept, and in the following section the



(a) Modulation of the mean



(b) Modes 2 & 3

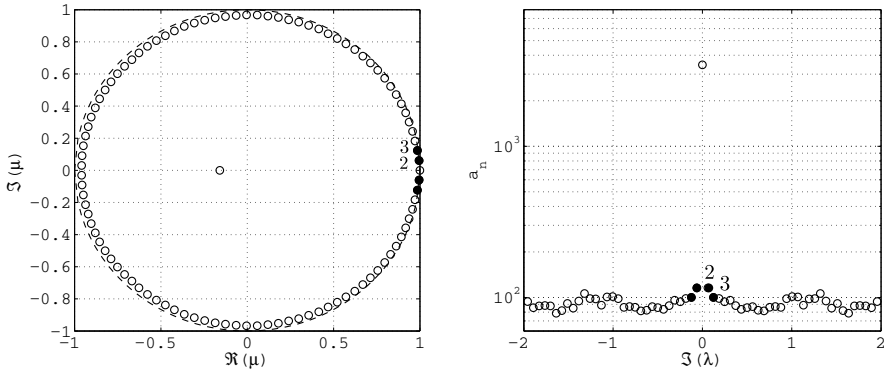
(c) Modes 4, 5, 6 & 7

Fig. 7: (a) Modulated mean flow profile using the six selected modes. (b,c) show the shape of the modes 2 & 3 and 4, 5, 6 & 7, respectively.

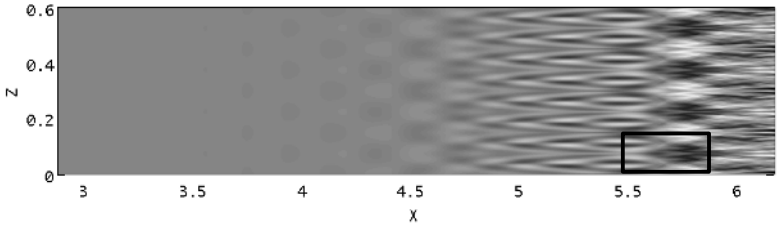
algorithm is tested on a large numerical data-set, where the parallelism is essential in order to perform the decomposition.

3.2 Transition to turbulence in a compressible flat-plate boundary layer

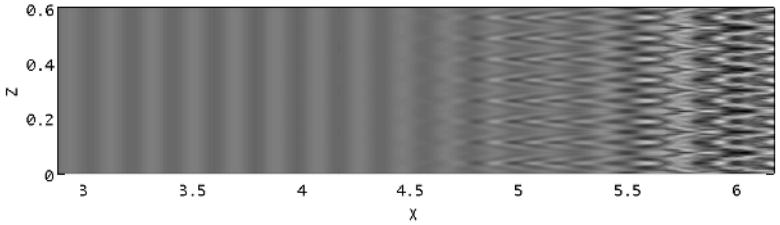
The dynamics of the pre-turbulent region of the controlled transitional boundary layer has previously been studied by [11]. Owing to the large size of the generated data-set and the goal to generate a reduced-order representation of the late-transitional regime, the decomposition of the transitional flow fields had to be limited to a rather small subregion, consisting of merely 0.3% of the full computational domain. Nonetheless, it could be demonstrated that a few low-frequency modes are able to recover the Reynolds shear-stress close to the wall. Even though the dynamics of the late-transitional regime had been captured by the study, there remained the question whether and how the identified dominant low-frequency modes connect to the fully turbulent regime further downstream. Using the parallel algorithmic extension, described above, and analyzing the full domain allows us now to address this question and to study and track low-frequency structures form the early laminar stage through transition into fully turbulent fluid motion.



(a) Eigenvalues (b) Spectrum



(c) x-z slice, Mode 2



(d) x-z slice, Mode 3

Fig. 8: Modal decomposition of the (fundamental) H-type transition. (a,b) DMD-spectrum and amplitude distribution, indicating by filled symbols the dominant low-frequency modes. (c,d) Wall-parallel slices of the streamwise velocity component. The selected DMD modes correspond to the Tollmien-Schlichting and subharmonic waves. The boxed region in (c) signifies the subdomain previously studied in [11].

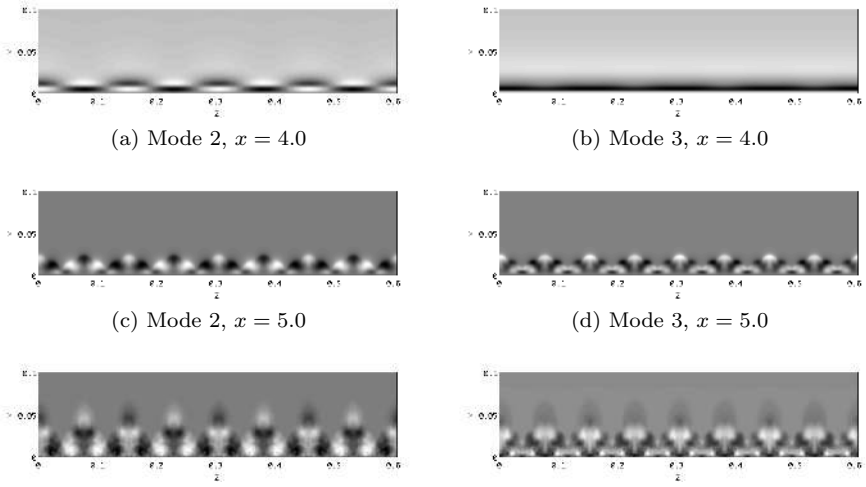
The domain used for applying the DMD algorithm consists of approximately half a billion grid points. Both controlled transitions of (fundamental) K-type and (subharmonic) H-type will be considered. The decomposition domain covers the full streamwise extent, from low-amplitude disturbances to nonlinear breakdown into the turbulent regime. The DMD modes of H-type transition, visualized by the

streamwise velocity component, are discussed first. Hundred and one equispaced snapshots are included in the data analysis; these snapshots span nearly two periods of the fundamental Tollmien-Schlichting wave. The eigenvalues of $\tilde{\mathbf{S}}$ from the dynamic mode decomposition are plotted in Figure 8(a). Two modes, mode 2 and mode 3, are highlighted in the figure and are selected for further analysis, as they describe a substantial part of the processed data sequence. The choice and amplitudes of these modes are determined using the optimization algorithm described in [21], and the resulting amplitude distribution is shown versus the detected frequencies in Figure 8(b). As demonstrated in [11], these two modes coincide with the fundamental Tollmien-Schlichting and subharmonic modes of the early transitional regime. The spatial evolution of the two DMD modes at a fixed wall-normal slicing plane is illustrated in Figure 8(c,d). The Tollmien-Schlichting wave is characterized by its two-dimensionality, which is illustrated in Figure 8(d) at the initial stage of the development of mode 2. However, as the flow approaches the nonlinear stage of the transition scenario, the two-dimensionality of the Tollmien-Schlichting wave breaks down, and the subharmonic wave exhibits a spanwise modification (or secondary instability) at the upstream portion of the domain. The spanwise extent of the domain is such that it comprises four wavelengths of the subharmonic wave, and this can be noticed in the extracted mode of Figure 8(c), especially in the streamwise region of $4.2 \leq x \leq 4.5$. The subdomain studied by [11] extended from $5.5 \leq x \leq 5.8$ and covered only one-fourth of the spanwise length (coinciding with one wave-length of the subharmonic wave); the size of the previously analyzed domain is indicated in Figure 8(c).

The shape of the modes extracted from our full data-set compares well in this region to the low-frequency modes of the earlier subdomain analysis. This suggests that the modes, representing the Tollmien-Schlichting wave and the subharmonic disturbances in the early transitional region, develop further downstream and connect to the low-frequency modes of the late-transitional regime studied in [11].

To gain a better understanding of the evolution of the modes in the downstream direction, we have extracted three cross-sectional slices at fixed streamwise locations. Comparison of the streamwise slices in Figure 9(a,b) illustrates the three-dimensional shape of the subharmonic disturbances versus the two-dimensional nature of the Tollmien-Schlichting wave. These modes are reminiscent of the modal shapes that are computed through linear stability analysis at this stage of the transition process, since the amplitude of the disturbances is still relatively low. As the modes evolve downstream, mode 2 maintains the spanwise wavelength of the initial subharmonic wave. This is illustrated by the shape of this mode, depicted in Figure 9(c) and the alternating sign along the spanwise direction. Mode 3 evolves in a different manner: no alternating sign is observed in the shape of the mode. Similarly, at the streamwise location $x = 6.0$, close to breakdown into turbulence, the overall shape of the modes still resembles the behavior observed farther upstream. Similar results can be observed by analyzing the (subharmonic) K-type transition, and the shape of the extracted dominant dynamic modes agree well with those presented in [11]. Now that we have identified the low-frequency modes of interest in both transitional scenarios, we can analyze the persistence of their shape within the whole domain, and well into the turbulent regime.

Comparing the development of these low-frequency modes into the turbulent regime for the two transition scenarios of K- and H-type also allows the assessment of the commonalities and differences these two scenarios and their principal struc-

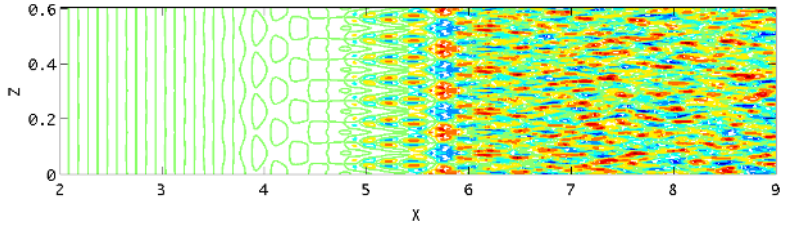


(a) Mode 2, $x = 4.0$ (b) Mode 3, $x = 4.0$

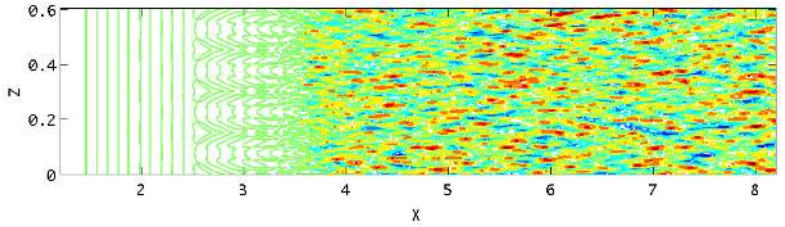
(c) Mode 2, $x = 5.0$ (d) Mode 3, $x = 5.0$

(e) Mode 2, $x = 6.0$ (f) Mode 3, $x = 6.0$

Fig. 9: Contours of the streamwise velocity of modes 2 and 3, for H-type transition.



(a) H-type transition normalized by x_0, where $Re|_{x_0} = 10^5$.



(b) K-type transition normalized by x_0, where $Re|_{x_0} = 10^5$

Fig. 10: Contours of the streamwise velocity showing the development of the lowest-frequency mode (mode 2) for the H- and K-type compressible boundary layer transition.

tures pose. To this end, the evolution of mode 2 is compared for the K- and H-type transition in Figure 10. K-type transition is characterized by the appearance of aligned Λ-shaped vortices in the late stages of transition, whereas these vortices appear in a staggered format during H-type transition. This characteristic is also reflected in the spatial shape of the DMD modes for the transitional regime. The trace of these Λ-vortices can be clearly seen in the low-frequency modes of the K-type transition at $x \approx 2.5$ and of the H-type transition at $x \approx 5.0$. These locations coincides with the streamwise location at which the skin-friction coefficient diverges from the laminar value as shown in [17]. The streamwise distance between the place where three-dimensional effects become visible to where the mode ceases to be spanwise periodic is markedly smaller in the K-type than in the H-type transition. This feature can be attributed mainly to the higher-amplitude initial disturbances that are introduced into the computational domain for the K-type transition, causing the region of nonlinear growth to shorten and the breakdown to turbulence to occur more rapidly. Although the spatial development of these modes depends on the specifics of the transition scenario, in the region where the flow is fully turbulent the differences between the two modes diminish [17]. In both transition scenarios the modes are predominantly composed of streaky structures in the streamwise direction. The shape, size and spacing of these streaks are similar to what has been reported for streaks in the buffer layer of turbulent wall-bounded flows. This is to be expected as the slices are taken from the buffer layer region of the turbulent boundary layer for both cases. These results suggest that low-frequency modes of dynamical importance in the late-transitional regime capture the streaky structures that characterize the buffer layer in the turbulent regime, and that these structures are independent of the transitional scenario preceding turbulence.

4 Summary and conclusions

A parallel QR algorithm for tall-and-skinny matrices has been embedded in common data-driven flow-field decomposition algorithms. The algorithm relies on a division of the data-matrix into submatrices along the row coordinate. The submatrices, sent to the available processors, are QR-decomposed locally, after which an additional QR-decomposition has to be performed and distributed to all processors, which determines the manner in which the local decompositions are assembled into the final result. The parallel version of the full decomposition allows the processing of very large data-sets, which could arise (i) naturally from large-scale simulations with many degrees of freedom or (ii) artificially, e.g., from composite data-sets or bifurcation studies where the embedding of multiple variables and/or parameter values inflates the state vector.

As numerical simulations for complex processes with a wide range of spatial and temporal scales become more and more commonplace, data sequences produced by these simulations dramatically increase in size. At the same time, decomposition and post-processing algorithms become evermore essential in the analysis of these large data-sets and in the reduction to a few coherent features and dominant processes contained in the snapshot sequence. The proposed algorithm and its variants scale well on a parallel architecture and can thus aid in the interpretation and analysis of large-scale fluid simulations.

Two applications have been included to illustrate the parallel algorithm and its advantages; both are transitional boundary layers. The first application treats a transitional boundary layer that exhibits a local separation bubble due to an imposed freestream pressure-gradient variation. The second application has been analyzed previously, though only in a very limited spatial region; this case has been included to assess the validity and prevalence of the earlier results when the far larger domain is treated by the parallel algorithm. In both cases, the transition region — spanning the laminar state, the rise of primary and secondary instabilities, and their ultimate breakdown — can be processed in its entirety which permits us to link upstream influences to downstream effects. In the case of the separated flow, the flapping of the shear layer and the dynamics of the reattachment region is captured using six low-frequency DMD modes. Furthermore, as far as the second transition scenario is concerned, the link between the dominant instability modes upstream and low-frequency streaky motion downstream in the turbulent region was illustrated by decomposing the entire domain.

Beyond the examples shown here, the parallel version of the data-driven decomposition techniques should appeal to a wide range of applications where massive data-sets need to be analyzed and the extraction of pertinent, but reduced information is critical for our understanding of the underlying flow physics.

Acknowledgements The authors would like to acknowledge the CTR summer program 2014 where the bulk of this work has been accomplished. In particular, Curtis Hamman is thanked for his support and insightful comments.

References

1. J. Demmel, L. Grigori, M. Hoemmen, J. Langou, SIAM J. Sci. Comput. **34**(1), 206 (2012)
2. K. Willcox, J. Peraire, AIAA Journal **40**(11), 2323 (2002)
3. M. Grilli, P.J. Schmid, S. Hickel, N.A. Adams, J. Fluid Mech. **700**, 16 (2012)
4. P.J. Schmid, D. Violato, F. Scarano, Experiments in Fluids **52**(6), 1567 (2012)
5. T.W. Muld, G. Efraimsson, D. Henningson, Flow, Turbulence and Combustion **88**(3), 279 (2012)
6. V. Statnikov, T. Sayadi, M. Meinke, P. Schmid, W. Schröder, Phys. Fluids **27**(1) (2015)
7. C.W. Rowley, Int. J. Bifurcation and Chaos **15**(3), 9971013 (2005)
8. I. Mezic, Annu. Rev. Fluid Mech. **45**(1), 357 (2013)
9. C.W. Rowley, I. Mezic, S. Bagheri, P. Schlatter, D.S. Henningson, J. Fluid Mech. **641**, 115 (2009)
10. P.J. Schmid, J. Fluid Mech. **656**, 5 (2010)
11. T. Sayadi, P.J. Schmid, J.W. Nichols, P. Moin, J. Fluid Mech. **748**, 278 (2014)
12. T. Sayadi, P.J. Schmid, F. Richecoeur, D. Durox, Phys. Fluids **27**(3), 037102 (2015)
13. J. Ye, Q. Li, H. Xiong, H. Park, R. Janardan, V. Kumar. IDR/QR: an incremental dimension reduction algorithm via QR decomposition (2005)
14. S. Perugini, M. Gonçalves, E.A. Fox, J. Intelligent Information Systems **23**(2), 107 (2004)
15. P. Luethi, C. Studer, S. Duetsch, E. Zgraggen, H. Kaeslin, N. Felber, W. Fichtner. Gram-Schmidt-based QR decomposition for MIMO detection: VLSI implementation and comparison (2008)
16. M. Heroux, R. Bartlett, V.H.R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, A. Williams, An overview of trilinos. Tech. Rep. SAND2003-2927, Sandia National Laboratories (2003)
17. T. Sayadi, C.W. Hamman, P. Moin, J. Fluid Mech. **724**, 480 (2013)
18. A.R. Benson, D.F. Gleich, J. Demmel, IEEE International Conference in Big Data (2013)
19. P. Spalart, M. Strelets, Journal of Fluid Mechanics **403**, 329 (2000)

Algorithm 1: finds the $\mathbf{Q}_i$ and $\mathbf{R}$ matrices

Input: $\mathbf{A}_i^{(1\to N)}$ data-matrix, dispersed on each processor
Output: $\mathbf{Q}_i$ and $\mathbf{R}$, the orthogonal and upper-triangular matrices
/* N_T: total number of data-points on each processor */
/* N_s : number of snapshots */
/* Index "i" indicates that the respective matrix or vector is defined on each processor, $i = 1, \cdots,$ nproc, where nproc is the total number of processors */
$\mathbf{A}_{\text{tmp}} = \mathbf{A}_i^{(1\to N)}$ // The data-matrix with snapshots from 1 to N
/* **Tau** is a square matrix of the size N_s */
/* *work* is a vector of the size "lwork" */
"lwork" $= N_T \times N_s$
CALL DGEQRF(N_T, N_s, $\mathbf{A}_{\text{tmp}}$, N_T, **Tau**, *work*, lwork, info)
for $j \leftarrow 1$ ***to*** N_s **do**
  $\mathbf{R}_i(1:j,j) = \mathbf{A}_{\text{tmp}}(1:j,j)$
end
/* Computing $\mathbf{Q}_{1i}$, which is stored in the rest of $\mathbf{A}_{\text{tmp}} \leftarrow \mathbf{Q}_{1i}$ */
CALL DORGQR(N_T, N_s, N_s, $\mathbf{A}_{\text{tmp}}$, N_T, **Tau**, *work*, lwork, info)
/* $\mathbf{R}_i$'s are gathered in $\mathbf{R}'$ in all the processors */
/* First $\mathbf{R}_i$ is stored in vector $tmpL_i$ */
for $j \leftarrow 1$ ***to*** N_s **do**
  $tmpL_i(1 + (j-1) * N_s : j * N_s) = \mathbf{R}_i(1:N_s, j)$
end
/* Gathering $tmpL_i$ to all processors in $tmpG$ */
CALL MPI_ALLGATHER($tmpL_i$, $N_s \times N_s$, MPI_REAL, $tmpG$, $N_s \times N_s$, MPI_REAL, MPI_COMM_WORLD, ierror) // $size(tmpG) = N_s \times N_s \times$ nproc
/* Forming $\mathbf{R}'$ from the vector $tmpG$ */
for $j \leftarrow 1$ ***to*** *nproc* **do**
  for $k \leftarrow 1$ ***to*** N_s **do**
    $\mathbf{R}'((N_s * (j-1) + 1) : (N_s * (j-1) + N_s), k) =$
    $tmpG((N_s \times N_s * (j-1) + (k-1) * N_s + 1) : (N_s \times N_s * (j-1) + k * N_s))$
  end
end
/* performing QR-factorization on $\mathbf{R}'$ */
"lwork" = N_s*nproc // "nproc" is the number of processors
/* *work* is a vector of the size lwork */
CALL DGEQRF(lwork, N_s, $\mathbf{R}'$, lwork, **Tau**, *work*, lwork, info)
/* Computing $\mathbf{R}$ */
for $i \leftarrow 1$ ***to*** N_s **do**
  $\mathbf{R}(1:i,i) = \mathbf{R}'(1:i,i)$
end
/* Computing $\mathbf{Q}_{2i}$, The orthogonal matrix is stored in the rest of $\mathbf{R}' \leftarrow \mathbf{Q}_{2i}$ */
CALL DORGQR(lwork, N_s, N_s, $\mathbf{R}'$, lwork, **Tau**, *work*, lwork, info)
/* Computing $\mathbf{Q}_i = \mathbf{A}_{\text{tmp}} \times \mathbf{R}'$ */
/* "myrank" is an integer value signifying the rank of the processor in the group */
CALL DGEMM ('N', 'N', N_T, N_s, N_s, 1.0, $\mathbf{A}_{\text{tmp}}$, N_T,
$\mathbf{R}'(1 + \text{myrank} \times N_s : N_s \times (\text{myrank}+1), :)$,N_T, 0.0, $\mathbf{Q}_i$, N_T)
return $\mathbf{Q}_i$, $\mathbf{R}$

Algorithm 2: finds the $\mathbf{U}_i$, $\mathbf{\Sigma}$ and $\mathbf{V}$ matrices

Input: $\mathbf{Q}_i$, dispersed on each processor and $\mathbf{R}$
Output: $\mathbf{U}_i$, $\mathbf{\Sigma}$, and $\mathbf{V}$
/* N_T: total number of data-points on each processor */
/* N_s : number of snapshots */
/* *work* is a vector of the size "lwork" */
"lwork" = $\max(1, 5N_s)$
/* Computing SVD of matrix $\mathbf{R}$ */
/* S, vector of the length N_s, storing the singular values */
CALL DGESVD('S', 'S', N_s, N_s, $\mathbf{R}$, N_s, S, $\mathbf{U}_R$, N_s, $\mathbf{V}$, N_s, *work*, lwork, info)
/* $\mathbf{\Sigma}$ is a square matrix of the size N_s */
for $i \leftarrow 1$ ***to*** N_s **do**
  $\mathbf{\Sigma}(i, i) = S(i)$
end
/* Computing $\mathbf{U}_i = \mathbf{Q}_i \times \mathbf{U}_R$ */
CALL DGEMM ('N', 'N', N_T, N_s, N_s, 1.0, $\mathbf{Q}_i$, N_T, $\mathbf{U}_R$, N_s, 0.0, $\mathbf{U}_i$, N_T)
return $\mathbf{U}_i$, $\mathbf{\Sigma}$, $\mathbf{V}$

Algorithm 3: finds the DMD modes, ϕ_i, and eigenvalues, μ

Input: $\mathbf{U}_i$, and $\mathbf{A}_i^{2\rightarrow(N+1)}$ dispersed on each processor, $\mathbf{\Sigma}$, and $\mathbf{V}$
Output: ϕ_i, and μ
/* N_T: total number of data-points on each processor */
/* N_s : number of snapshots */
/* Computing $\mathbf{B} = \mathbf{U}_i \times \mathbf{A}_i^{2\rightarrow(N+1)}$ on each processor */
/* $\mathbf{B}$ is a square matrix of the size N_s */
CALL DGEMM ('T', 'N', N_s, N_s, N_T, 1.0, $\mathbf{U}_i$, N_T, $\mathbf{A}_i^{2\rightarrow(N+1)}$, N_T, 0.0, $\mathbf{B}$, N_s)
/* Storing $\mathbf{B}$, in vector form, in *tmpL* */
for $j \leftarrow 1$ ***to*** N_s **do**
  $tmpL_i(1 + (j-1) * N_s : j * N_s) = \mathbf{B}(1 : N_s, j)$
end
/* Summing $tmpL_i$ across all processors and storing the result in *tmpG* */
CALL MPI_ALLREDUCE($tmpL$, $tmpG$, $N_s \times N_s$, MPI_REAL, MPI_SUM, MPI_COMM_WORLD) // $size(tmpG) = N_s \times N_s$
/* Forming $\mathbf{C}$ from the vector *tmpG* */
for $j \leftarrow 1$ ***to*** N_s **do**
  $\mathbf{C}(1 : N_s, j) = tmpG(1 + (j-1) * N_s : j * N_s)$
end
/* Computing $\mathbf{B} = \mathbf{C} \times \mathbf{V}^T$ */
CALL DGEMM ('N', 'T', N_s, N_s, N_s, 1.0, $\mathbf{C}$, N_s, $\mathbf{V}$, N_s, 0.0, $\mathbf{B}$, N_s)
/* Computing eigenvalues μ_i, μ_r and eigenvectors $\mathbf{Y}$ of $\tilde{\mathbf{S}} = \mathbf{B} \times \mathbf{\Sigma}$ */
/* μ_i, μ_r and β, vectors of the size N_s */
/* $\mathbf{Y}$ is a matrix of the size N_s */
/* *work* is a vector of the size "lwork" */
"lwork" = $\max(1, 8N_s)$
CALL DGGEV('N', 'V', N_s, $\mathbf{B}$, N_s, $\mathbf{\Sigma}$, N_s, μ_i, μ_r, β, $\mathbf{B}$, 1, $\mathbf{Y}$, N_s, *work*, lwork, info)
for $i \leftarrow 1$ ***to*** N_s **do**
  $\mathbf{Y}(i, :) = \mathbf{Y}(i, :) \times \mathbf{\Sigma}(i, i)$
end
/* Eigenvalues: */
$\mu_r = \mu_r / \beta$
$\mu_i = \mu_i / \beta$
/* Computing $\phi_i = \mathbf{U}_i \times \mathbf{Y}$ */
CALL DGEMM('N', 'N', N_T, N_s, N_s, 1.0, $\mathbf{U}_i$, N_T, $\mathbf{Y}$, N_s, 0.0, ϕ, N_T)
return ϕ_i, μ

20. F. Cadieux, J.A. Domaradzki, T. Sayadi, S. Bose, J Fluids Engineering **136**(6), 60902 (2014)
21. M.R. Jovanović, P.J. Schmid, J.W. Nichols, Phys. Fluids **26**(2), 024103 (22 pages) (2014)