Journal of Computational and Applied Mathematics 307 (2016) 374–384

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Approximate partitioned method of snapshots for POD

Zhu Wang [a,*], Brian McBee [b], Traian Iliescu [c]

[a] Department of Mathematics, University of South Carolina, 1523 Greene Street, Columbia, SC 29208, USA

[b] Department of Mathematics and Statistics, Air Force Institute of Technology, 2950 Hobson Way, Wright-Patterson Air Force Base, OH 45433, USA

[c] Department of Mathematics, Virginia Tech, 225 Stanger Street, Blacksburg, VA 24061, USA

ARTICLE INFO

Article history:
Received 10 April 2015
Received in revised form 21 November 2015

Keywords:
Proper orthogonal decomposition
Domain decomposition
Partitioned singular value decomposition
Partitioned method of snapshots

ABSTRACT

The proper orthogonal decomposition (POD) method has been widely used to construct efficient numerical surrogate models for computationally intensive applications in control and optimization. An inherent challenge with this method is that POD basis generation can be computationally expensive due to the huge size of the input snapshot data obtained from typical high-fidelity, large-scale dynamic system simulations. However, if the process can be distributed into much smaller tasks over multiple processors in parallel, computational time can be drastically reduced. In this paper, we put forth a novel partitioned method for generating the POD basis from snapshot data. This method preserves the distributed nature of the data and takes advantage of parallelism for computation. Additionally, it greatly reduces subtask communication volume. Two numerical examples are presented that demonstrate the effectiveness of the new method.

Published by Elsevier B.V.

1. Introduction

The requirement for repeated numerical simulations of large-scale dynamical systems presents immense challenges in many scientific and engineering applications. To alleviate the tremendous computational cost of such numerical simulations, the proper orthogonal decomposition (POD) method has seen wide use to produce computationally efficient reduced-order models (ROMs). The method extracts a set of POD basis functions from given data (a so-called snapshot matrix), and uses a small subset of leading basis functions to construct state variable approximations.

The snapshot data feeding a POD basis are usually collected from one or several runs of high-fidelity numerical simulations of the original system. Due to the complexity of such a system, the number of degrees of freedom (DOF) in the simulations can be extremely large. For instance, to resolve the entire spatio-temporal range of scales in 3D turbulent flows, $\mathcal{O}(\mathrm{Re}^{9/4})$ grid points are needed, where Re is the Reynolds number [1]. Hence, the number of DOF is already on the order of millions even for a moderate $\mathrm{Re} = 10^3$. Therefore, parallel programs are usually designed to break such a large-scale problem into discrete "chunks" of work and allocate them to multiple tasks, which can be implemented simultaneously in a multiprocessor environment. As one of the most popular ways to partition the work, the domain decomposition method (DDM) has been widely studied [2,3]. Instead of solving the original problem directly on the whole domain, the DDM considers modified problems over small (overlapping or non-overlapping) subdomains. Each subdomain problem runs separately and only small-scale problems with few unknowns per subdomain are used to coordinate solutions among the adjacent subdomains. In other words, this method allows each processor of a parallel computer to independently/simultaneously

* Corresponding author.
E-mail address: wangzhu@math.sc.edu (Z. Wang).

http://dx.doi.org/10.1016/j.cam.2015.11.023
0377-0427/Published by Elsevier B.V.

handle individual subdomains and integrate in time locally. Once the time integration is complete, the simulation data is then distributed over the multiprocessors. To fully represent the typical complex dynamical system, tens of thousands of snapshots must be selected. For a parametric system, the number of snapshots can be much larger in order to describe various behavior of the system as the physical parameters vary [4]. As a result, after the numerical simulations, the final snapshot data, distributed across multiple processors, can become unmanageably vast.

The POD basis typically comprises the left singular vectors of the snapshot matrix, which can be obtained by singular value decomposition (SVD) [5]. When the snapshot matrix is tall and skinny (the number of DOF is much greater than that of selected snapshots), the method of snapshots (MOS) is an efficient alternative way to compute the POD basis [6]. This method, instead of computing an SVD of the snapshot matrix, computes the eigenvectors of the snapshot covariance matrix. The POD basis is then calculated by post-multiplying the snapshot matrix with these eigenvectors.

In the domain decomposition setting, both the SVD and MOS can become computationally impractical (if not impossible) for generating POD basis functions of complex systems because of memory management issues. First, assembly of the snapshots requires heavy communication between the processor managing aggregation and the subdomain processors. Second, the large size and density of the resulting snapshot matrix can cause these methods to burden a single processor with an excessively heavy load. Therefore, several algorithms have been developed to alleviate these difficulties.

At present, several parallel SVD algorithms have been developed [7,8], but their utility is limited to sparse matrices. A partitioned SVD approach was proposed in [9], which first extracts a small number of dominant right singular vectors from each local data set (i.e. subdomain). These vectors are aggregated from each processor to produce a small matrix on a single processor. The right singular vectors of this new matrix are used to approximate those of the original snapshot matrix based on a hypothesis that right singular vectors of the local data tend to stay the same as those of the full snapshot matrix, which is questionable in general. To remove this discrepancy, a filtered subspace iteration is used in [9]. However, this iterative algorithm still introduces a step that applies SVD on a global matrix (with the same number of rows as DOF in the snapshot matrix).

In this work, we propose a new partitioned method of snapshots for the POD computation, which preserves the distributed nature of the snapshot data in the multiprocessor environment and requires low communication overhead. Compared with the partitioned SVD algorithm in [9], the new method does not rely on any assumptions of a relationship between local singular vectors and global singular vectors, and still yields an accurate POD basis approximation.

The rest of this paper is organized as follows: In Section 2, we summarize the existing tools for generating the POD basis and propose a new partitioned method of snapshots. Numerical tests are presented in Section 3, where two sample problems (Burgers equation and gravity current) are considered to verify the effectiveness of the new method. Concluding remarks are made in Section 4.

2. The POD basis

Suppose the numerical simulation of a large-scale dynamical system has been executed by parallel computing in a distributed memory computer system. The partition of computational work is based on a domain decomposition and the resulting subdomain information is allocated over p processors. The system is integrated in time locally on each processor during the simulations. At the end of the time integration, the snapshot matrix remains distributed over the p processors.

Define by $\mathbf{S}_i$ the $n_i \times m$ matrix representing the time history of the state variables on the ith processor, where n_i is the number of local DOF and m is the number of snapshots. We assume each DOF is native to exactly one processor, i.e., no overlap DOF is shared among processors. If an overlapping domain decomposition method is considered, a straightforward mapping can be applied to extract $\mathbf{S}_i$ from the local data. The full snapshot matrix $\mathbf{S}$ can be assembled from the local matrices, that is,

$$\mathbf{S} = \left[\mathbf{S}_1^\intercal, \mathbf{S}_2^\intercal, \ldots, \mathbf{S}_p^\intercal\right]^\intercal,$$

whose shape is $n \times m$ with $n = \sum_{i=1}^{p} n_i$.

Given the snapshot matrix, the POD method seeks a low-dimensional basis that could be used to well-approximate the state variables. POD basis functions are typically the left singular vectors of the matrix [10], which can be computed by SVD directly. However, with the data distributed over multiple processors, a brute force approach requires moving all the local data onto a single processor to assemble the large, dense snapshot matrix and perform the SVD on it.

In distributed memory parallel computers, distinct memory is allocated to each processor which has fast access only to its own local memory. Communication is required for a processor to access data stored on another processor's local memory. Such communication takes $s + r_a n$ time, where s is the latency, n is the number of bytes being transferred, and r_a is the incremental time per 1 byte. For example, in a typical workstation cluster, $s = 950\ \mu$s and $r_a = 7\ \mu$s/word (see [11, Table 2.1]). Communication cost can be even greater when emerging architectures, e.g. GPUs, are employed, where there is a significant cost in communicating data between the host and device. Indeed, any process that requires massive processor–processor communication creates a heavy time burden.

Therefore, efficient algorithms need to be developed that avoid such dramatic computational cost and communication effort to generate the POD basis. To this end, we develop a new partitioned method of snapshots, which takes advantage of the parallelism of the simulations. In the following, we first review several existing methods, then present a new approach.

For comparison, two criteria are considered: (i) computational complexity in terms of floating-point operations (flops); and (ii) communication effort in terms of floating points to be transferred.

2.1. Existing techniques

Singular value decomposition [12]. In general, when the matrix size is small, one can use SVD directly for generating the POD basis. The SVD of the snapshot matrix $\mathbf{S}$ is expressed as

$$\mathbf{S} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathsf{T}},$$

where $\mathbf{U}$, $\mathbf{V}$ are unitary and $\boldsymbol{\Sigma}$ is diagonal. The POD basis is composed of column vectors, where the jth basis function $\boldsymbol{\varphi}_j$ is the jth column of $\mathbf{U}$.

Method of snapshots [6]. In most cases, the snapshot matrix is tall and skinny ($m \ll n$) representing many more DOF than snapshots. Instead of applying SVD directly (which is computationally expensive), one can utilize the method of snapshots. This method manipulates a small eigenvalue problem first as follows:

$$\mathbf{S}^{\mathsf{T}}\mathbf{S}\,\mathbf{z}_j = \lambda_j\,\mathbf{z}_j, \quad \text{for } j = 1, \ldots, r, \tag{1}$$

where $\mathbf{S}^{\mathsf{T}}\mathbf{S}$ is known as the snapshot covariance matrix with size $m \times m$. Note that

$$\mathbf{S}^{\mathsf{T}}\mathbf{S} = \mathbf{V}\boldsymbol{\Sigma}^2\mathbf{V}^{\mathsf{T}} \Leftrightarrow (\mathbf{S}^{\mathsf{T}}\mathbf{S})\,\mathbf{V} = \mathbf{V}\boldsymbol{\Sigma}^2. \tag{2}$$

Comparing (2) with (1), we see that the right singular vectors of $\mathbf{S}$ are the same as the eigenvectors of $\mathbf{S}^{\mathsf{T}}\mathbf{S}$; and the singular values of $\mathbf{S}$ are the positive square roots of the eigenvalues of $\mathbf{S}^{\mathsf{T}}\mathbf{S}$. Once the eigenvector $\mathbf{z}_j$ in (1) is obtained, the jth POD basis function $\boldsymbol{\varphi}_j$ is determined by

$$\boldsymbol{\varphi}_j = \frac{1}{\sqrt{\lambda_j}} \sum_{\ell=1}^{m} (\mathbf{z}_j)_\ell\, \mathbf{S}_{(\cdot,\ell)}, \quad 1 \le j \le r, \tag{3}$$

where $(\mathbf{z}_j)_\ell$ is the ℓ-th component of the jth eigenvector $\mathbf{z}_j$ of $\mathbf{S}^{\mathsf{T}}\mathbf{S}$ and $\mathbf{S}_{(\cdot,\ell)}$ is the ℓ-th column of $\mathbf{S}$.

Partitioned singular value decomposition [9]. Based on the assumption that right singular vectors of the local data tend to stay the same as those of the full snapshot matrix, the partitioned singular value decomposition (PSVD) method was developed by Beattie et al. in [9]. First, instead of assembling the full snapshot matrix, their method computes an SVD of local data $\mathbf{S}_i$ on each processor, that is,

$$\mathbf{S}_i = \mathbf{U}_i\boldsymbol{\Sigma}_i\mathbf{V}_i^{\mathsf{T}}. \tag{4}$$

A matrix $\boldsymbol{\mathcal{V}}$ is then constructed by putting together leading right singular vectors of $\mathbf{S}_i$ over all processors

$$\boldsymbol{\mathcal{V}} = \left[\mathbf{V}_1^q, \mathbf{V}_2^q, \ldots, \mathbf{V}_p^q\right],$$

where $\mathbf{V}_i^q = \mathbf{V}_{i(\cdot,1:q)}$ consists of q dominant right singular vectors. It was argued in [9] that $\boldsymbol{\mathcal{V}}$ is a good sample set for the right singular vectors of the full data $\mathbf{S}$, since it contains the leading right singular vectors on all subdomains. Therefore, the left singular vectors of $\boldsymbol{\mathcal{V}}$, $\widehat{\mathbf{V}}$, can be regarded as an approximation of $\mathbf{V}$. Note that because the size of $\boldsymbol{\mathcal{V}}$ is small, $m \times pq$, the SVD of $\boldsymbol{\mathcal{V}}$ can be completed at a low computational cost. Take the first r columns of $\widehat{\mathbf{V}}$, $\widehat{\mathbf{V}}^r$ with $q \le r \le 2q$. The POD basis $\widehat{\mathbf{U}}$ is approximated by the left singular vectors of $\mathbf{S}\widehat{\mathbf{V}}^r$.

Algorithm 1: Partitioned Singular Value Decomposition (Non-iterative)

Let $\mathbf{S}_i$ be local data on the ith processor.
for $i = 1$ **to** p **do**
  $[\mathbf{U}_i, \boldsymbol{\Sigma}_i, \mathbf{V}_i]$ = svd($\mathbf{S}_i$) locally;
  put $\mathbf{V}_i^q$ in $\boldsymbol{\mathcal{V}}$;
end
do $[\widehat{\mathbf{V}}, \sim, \sim]$ = svd($\boldsymbol{\mathcal{V}}$);
take $\widehat{\mathbf{V}}^r = \widehat{\mathbf{V}}(\cdot,\ 1:r)$;
calculate $\mathbf{S}\widehat{\mathbf{V}}^r = [\mathbf{S}_1\widehat{\mathbf{V}}^r, \ldots, \mathbf{S}_p\widehat{\mathbf{V}}^r]$ locally;
do $[\widehat{\mathbf{U}}, \sim, \sim]$ = svd($\mathbf{S}\widehat{\mathbf{V}}^r$).

Since in general there exist discrepancies between the right singular vectors of the local data and those of the full snapshot data, in order to improve the method, Beattie et al. designed in [9] a filtered subspace iteration algorithm.

Partitioned method of snapshots. Consider the snapshot covariance matrix

$$\mathbf{S}^{\mathsf{T}}\mathbf{S} = \sum_{i=1}^{p} \mathbf{S}_i^{\mathsf{T}}\mathbf{S}_i. \tag{5}$$

Taking advantage of the parallelism of the simulations, to generate this matrix, one can first compute $\mathbf{D}_i = \mathbf{S}_i^{\mathsf{T}}\mathbf{S}_i$ on each processor, then transfer $\mathbf{D}_i$ to a single processor for addition followed by solution of the eigenvalue problem (1). The components of POD basis functions on each subdomain are computed by vector multiplications of the eigenvectors $\mathbf{V}$ and the local data $\mathbf{S}_i$ on each processor:

$$\boldsymbol{\varphi}_j^i(\cdot) = \frac{1}{\sqrt{\lambda_j}} \sum_{\ell=1}^{m} (\mathbf{v}_j)_\ell \, \mathbf{S}_{i,(\cdot,\ell)}, \quad 1 \le j \le r, \ 1 \le i \le p. \tag{6}$$

The process is outlined in Algorithm 2.

Algorithm 2: Partitioned Method of Snapshots (PMOS)

Let $\mathbf{S}_i$ be local data on the ith processor.
for $i = 1$ **to** p **do**
  Evaluate $\mathbf{D}_i = \mathbf{S}_i^{\mathsf{T}}\mathbf{S}_i$ locally;
end
do $\mathbf{D} = \sum_{i=1}^{p} \mathbf{D}_i$;
do $[\mathbf{V}, \boldsymbol{\Sigma}] = \mathrm{eig}(\mathbf{D})$;
choose r s.t. $1 - \sum_{i=1}^{r} \lambda_i / \sum_{i=1}^{d} \lambda_i < \epsilon_1$;
for $i = 1$ **to** p **do**
  for $j = 1$ **to** r **do**
    calculate POD basis functions $\boldsymbol{\varphi}_j^i = \frac{1}{\sqrt{\lambda_j}} \mathbf{S}_i \mathbf{V}_{(\cdot,j)}$ locally;
  end
end

This partitioned method yields the same basis functions as the standard MOS but makes use of the distributed nature of the snapshots so that some dot products or matrix–vector products can be performed in parallel. This idea is common in modern linear algebra frameworks such as PETSc or Trilinos and has been implemented in several open-source applications, for example, Feel++ and rb00mit.

Comparisons among existing approaches. To measure computational cost of the preceding algorithms, we compare the number of flops needed in each process. In this study, we denote by *flop* a single elementary floating-point operation including sum, subtraction, multiplication or division. We also express complexity by ignoring low-order terms when counting flops in a given algorithm.

Assumption 2.1. Assume the following relationships hold for POD basis number r, subdomain number p, and local DOF number n_i:

$$rp \ll \sum_{i=1}^{p} n_i \quad \text{and} \quad (rp)^2 \ll \sum_{i=1}^{p} n_i^2.$$

It is known that computing the SVD of an $n \times m$ matrix has complexity $\mathcal{O}(n^2m + nm^2 + m^3)$ (see, e.g., [12,13]). The computational complexity of MOS is $\mathcal{O}(nm^2 + rnm + m^3)$, where $\mathcal{O}(nm^2)$ flops are needed for evaluating $\mathbf{S}^{\mathsf{T}}\mathbf{S}$, $\mathcal{O}(m^3)$ flops for the eigen-decomposition of $\mathbf{S}^{\mathsf{T}}\mathbf{S}$ (see, e.g., [12,14]) and $\mathcal{O}(rnm)$ flops for the calculation of the POD basis by Eq. (3). For the PSVD, we only estimate the non-iterative version (Algorithm 1). Note that in order to improve POD accuracy for PSVD, the iterative method must be utilized [9], which increases the computational cost. The computational complexity of the non-iterative algorithm is $\mathcal{O}\left(\sum_{i=1}^{p}(n_i^2m + n_im^2 + m^3 + n_irm)\right) + \mathcal{O}\left(r^3p^3 + n^2r + nr^2 + r^3\right)$, where $q \le r \le 2q$: the execution of SVD on local matrices $\mathbf{S}_i$ in parallel requires $\mathcal{O}\left(\sum_{i=1}^{p}(n_i^2m + n_im^2 + m^3)\right)$ flops in total, the computational cost of SVD on $\mathcal{V}$ is $\mathcal{O}\left(m^2rp + mr^2p^2 + r^3p^3\right)$, the matrix multiplication for $\widehat{\mathbf{SV}}^r$ needs $\mathcal{O}\left(\sum_{i=1}^{p} n_irm\right)$ flops, and the SVD of $\widehat{\mathbf{SV}}^r$ on a single processor at the last step of the algorithm needs $\mathcal{O}\left(n^2r + nr^2 + r^3\right)$ flops. Note that we ignore $\mathcal{O}\left(m^2rp + mr^2p^2\right)$ in the total complexity based on Assumption 2.1. In each of the above cases, the existing algorithm leads to computational complexity proportional to the number of DOF or even the square of the number of DOF. Much of the PMOS process has similar computational complexity to the standard MOS. For the first step, evaluating $\mathbf{D}_i = \mathbf{S}_i^{\mathsf{T}}\mathbf{S}_i$ has complexity $\mathcal{O}(\sum_{i=1}^{p} n_im^2)$ and $\mathbf{D} = \sum_{i=1}^{p} \mathbf{D}_i$ has complexity $\mathcal{O}((p-1)m^2)$. For the second step, evaluating the eigenvalue decomposition of $\mathbf{D}$ has

complexity $\mathcal{O}(m^3)$. In the third step, calculating the POD basis has complexity $\mathcal{O}(\sum_{i=1}^{p} n_i rm)$. Note that $p \ll \sum_{i=1}^{p} n_i$ by Assumption 2.1 and we have complexity $\mathcal{O}(\sum_{i=1}^{p} n_i m^2 + \sum_{i=1}^{p} n_i rm + m^3)$. Despite such similarities, the first and last steps of the PMOS are implemented locally on multiple processors for significant savings in processing time.

To estimate the communication effort in the SVD, MOS and PSVD algorithms, we count the number of floating points in the matrices to be transferred among multiple processors.

In SVD and MOS, the full snapshot matrix must be assembled. Because the snapshot matrix $\mathbf{S}_{n\times m}$ is large and dense, the assembling process is time consuming. For example, for a complex system with 10^6 DOF and 10 distinct parameters, if 10^3 snapshots are collected for each parameter, then the size of the snapshot matrix will be $10^6 \times 10^4$. If entries of the matrix are double-precision floating points in the computer system, the amount of data transferred to assemble the matrix is approximately 70 GB. The non-iterative PSVD method requires the dominant right singular vectors $\mathbf{V}_i^q$, the dominant left singular vectors $\widehat{\mathbf{V}}^r$ and the matrices $\mathbf{S}_i\widehat{\mathbf{V}}^r$ to be transferred among multiple processors. The total number of entries in these matrices is $nr + mr + mpq$. When $r = 50$, the size of this data is approximately 3.7 GB. When iterative PSVD is used, the amount of data transferred increases significantly. While the standard MOS requires that all local snapshot matrices $\mathbf{S}_i$ be moved to a single processor, PMOS only requires $\mathbf{D}_i$ and $\mathbf{V}^r$ to be transferred among processors. Furthermore, the total size of $\mathbf{D}_i$ and $\mathbf{V}^r$ is $pm^2 + mr$, which is much smaller than the size of all local matrices, nm in the typical case of very large number of DOF n. Therefore, the PMOS is often more efficient in processor-to-processor communication than the standard MOS. However, when the number of snapshots m becomes large (on order of n), this method still imposes a heavy communication burden (on order of m^2 flops for fixed r and p).

In summary, when DOF and snapshot counts are large, the existing algorithms are burdened with either high computational complexities or high-volume data communications. Since the efficiency of computations and communications in POD basis generation has a significant impact on the applications of ROMs in the domain decomposition setting, we propose a novel partitioned method of snapshots, which reduces data transfer beyond PMOS (Algorithm 2) while achieving desirable accuracy.

2.2. Approximate partitioned method of snapshots

This approach is motivated by the following observations: Based on the SVD of a local snapshot matrix $\mathbf{S}_i$, the covariance matrix can be expressed as

$$\begin{aligned}\mathbf{S}^\intercal\mathbf{S} &\overset{(5)}{=} \sum_{i=1}^{p} \mathbf{V}_i\boldsymbol{\Sigma}_i^2\mathbf{V}_i^\intercal \\ &= \left[\mathbf{V}_1\boldsymbol{\Sigma}_1^\intercal, \mathbf{V}_2\boldsymbol{\Sigma}_2^\intercal, \ldots, \mathbf{V}_p\boldsymbol{\Sigma}_p^\intercal\right]\begin{bmatrix}\boldsymbol{\Sigma}_1\mathbf{V}_1^\intercal \\ \boldsymbol{\Sigma}_2\mathbf{V}_2^\intercal \\ \vdots \\ \boldsymbol{\Sigma}_p\mathbf{V}_p^\intercal\end{bmatrix} \\ &= \mathbf{W}\mathbf{W}^\intercal, \end{aligned} \tag{7}$$

where

$$\mathbf{W} = \left[\mathbf{V}_1\boldsymbol{\Sigma}_1^\intercal, \mathbf{V}_2\boldsymbol{\Sigma}_2^\intercal, \ldots, \mathbf{V}_p\boldsymbol{\Sigma}_p^\intercal\right].$$

Clearly, the eigenvectors of $\mathbf{S}^\intercal\mathbf{S}$ coincide with the left singular vectors of $\mathbf{W}$. Thus, we propose a new approximate partitioned method of snapshots, outlined in Algorithm 3.

Here the SVD of local data $\mathbf{S}_i$ is first performed on each processor and r_i is chosen such that the singular value $\sigma_i^{r_i+1}$ is less than a prescribed tolerance ϵ_0, i.e.,

$$\sigma_i^{r_i+1} < \epsilon_0, \quad \text{for } i = 1, \ldots, p.$$

Let $\mathbf{V}_i^{r_i}$ be the first r_i columns of $\mathbf{V}_i$ and $\boldsymbol{\Sigma}_i^{r_i}$ be the upper-left $r_i \times r_i$ block matrix of $\boldsymbol{\Sigma}_i$. Define

$$\mathbf{W}^r = \left[\mathbf{V}_1^{r_1}(\boldsymbol{\Sigma}_1^{r_1})^\intercal, \mathbf{V}_2^{r_2}(\boldsymbol{\Sigma}_2^{r_2})^\intercal, \ldots, \mathbf{V}_p^{r_p}(\boldsymbol{\Sigma}_p^{r_p})^\intercal\right].$$

Note that $\mathbf{W}^r(\mathbf{W}^r)^\intercal = \sum_{i=1}^{p} \mathbf{V}_i^{r_i}(\boldsymbol{\Sigma}_i^{r_i})^2(\mathbf{V}_i^{r_i})^\intercal$ is a good approximation of $\mathbf{W}\mathbf{W}^\intercal$ for a small tolerance ϵ_0. Thus, its eigenvectors and eigenvalues emerge as good approximations of the eigenvectors and eigenvalues of $\mathbf{W}\mathbf{W}^\intercal$ and $\mathbf{S}^\intercal\mathbf{S}$. The eigenvectors of $\mathbf{W}^r(\mathbf{W}^r)^\intercal$ can be obtained by taking the SVD of $\mathbf{W}^r$,

$$\mathbf{W}^r = \mathbf{X}\boldsymbol{\Lambda}\mathbf{Y}^\intercal,$$

where $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_d)$. The right singular vectors and singular values of the full snapshot matrix $\mathbf{S}$ can then be approximated by

$$\mathbf{V} \approx \mathbf{X} \quad \text{and} \quad \boldsymbol{\Sigma} \approx \boldsymbol{\Lambda}.$$

Algorithm 3: Approximate Partitioned Method of Snapshots (APMOS)

Let $\mathbf{S}_i$ be local data on the ith processor.
for $i = 1$ **to** p **do**
  $[\mathbf{U}_i, \mathbf{\Sigma}_i, \mathbf{V}_i]$ = svd($\mathbf{S}_i$) locally;
  select r_i, s.t., $\sigma_i^{r_i+1} < \epsilon_0$;
  take $\mathbf{V}_i^{r_i} = \mathbf{V}_{i,\,(\cdot,\,1:r_i)}$ and $\mathbf{\Sigma}_i^{r_i} = \mathbf{\Sigma}_{i,\,(1:r_i,\,1:r_i)}$;
end
assemble $\mathbf{W}^r = \left[\mathbf{V}_1^{r_1}(\mathbf{\Sigma}_1^{r_1})^\intercal, \ldots, \mathbf{V}_p^{r_p}(\mathbf{\Sigma}_p^{r_p})^\intercal\right]$;
do $[\mathbf{X}, \mathbf{\Lambda}, \mathbf{Y}]$ = svd($\mathbf{W}^r$);
choose r, s.t. $1 - \sum_{i=1}^{r}\lambda_i / \sum_{i=1}^{d}\lambda_i < \epsilon_1$;
for $i = 1$ **to** p **do**
  for $j = 1$ **to** r **do**
    calculate POD basis functions $\widetilde{\boldsymbol{\varphi}}_j^i = \frac{1}{\sqrt{\lambda_j}}\mathbf{S}_i\mathbf{X}_{(\cdot,j)}$ locally;
  end
end

The components of POD basis functions on each subdomain are computed by vector multiplications of the left singular vectors in $\mathbf{X}$ and the local data $\mathbf{S}_i$ on each processor,

$$\widetilde{\boldsymbol{\varphi}}_j^i(\cdot) = \frac{1}{\sqrt{\lambda_j}}\sum_{\ell=1}^{m}(\mathbf{x}_j)_\ell\,\mathbf{S}_{i,(\cdot,\ell)}, \quad 1 \le j \le r,\ 1 \le i \le p. \tag{8}$$

This method provides an opportunity for nearly end-to-end distributed/parallel data processing. Each processor performs an SVD on its local data. Only a small amount of data must be transferred from the local data processors to a single processor for SVD computation of a relatively small matrix, $\mathbf{W}^r$. The final POD basis components are then calculated by local matrix/vector multiplication.

Assumption 2.2. Assume the following relationships hold:

$$\sum_{i=1}^{p} r_i \ll \sum_{i=1}^{p} n_i, \qquad \sum_{i=1}^{p} r_i^2 \ll \sum_{i=1}^{p} n_i^2, \quad \text{and} \quad \left(\sum_{i=1}^{p} r_i\right)^2 \ll \sum_{i=1}^{p} n_i^2.$$

In Algorithm 3, the first step requires an SVD of local data $\mathbf{S}_i$ with complexity $\mathcal{O}(\sum_{i=1}^{p}(n_i^2 m + n_i m^2 + m^3))$. Then $\mathbf{V}_i^{r_i}\mathbf{\Sigma}_i^{r_i}$ is evaluated with $\mathcal{O}(\sum_{i=1}^{p} m r_i^2)$ flops. The second step requires an SVD of $\mathbf{W}^r$ with the complexity $\mathcal{O}(m^2\sum_{i=1}^{p} r_i + m(\sum_{i=1}^{p} r_i)^2 + (\sum_{i=1}^{p} r_i)^3)$. The third step computes the POD basis components with cost $\mathcal{O}(\sum_{i=1}^{p} n_i r m)$. Based on Assumption 2.2, we may ignore $\mathcal{O}(\sum_{i=1}^{p} m r_i^2 + m^2\sum_{i=1}^{p} r_i + m(\sum_{i=1}^{p} r_i)^2)$ in the total computational cost leaving the complexity of Algorithm 3 to be $\mathcal{O}(\sum_{i=1}^{p}(n_i^2 m + n_i m^2 + m^3 + n_i r m) + \mathcal{O}(\sum_{i=1}^{p} r_i)^3)$. Furthermore, as discussed, the first and last steps of this algorithm are executed in parallel.

The dominant communication cost for this approach comes from the assembly of $\mathbf{W}^r$, where the matrix $\mathbf{V}_i^{r_i}(\mathbf{\Sigma}_i^{r_i})^\intercal$ is sent from each ith local processor to a single processor, and an $m \times r$ matrix X^r along with an $r \times r$ diagonal matrix Λ^r are transferred back to each processor. The total size of these matrices is $m\sum_{i=1}^{p} r_i + mr + r$.

Note that, compared with Algorithm 2, the number of floating points to be transferred is reduced to $\mathcal{O}(m\sum_{i=1}^{p} r_i)$ from $\mathcal{O}(m^2)$. When m is large as in many ROMs for complex systems, this new partitioned method of snapshots will greatly decrease the communication cost because $\sum_{i=1}^{p} r_i$ is often much smaller than m.

Remark 2.1. When computing $\mathbf{V}_i^{r_i}\mathbf{\Sigma}_i^{r_i}$, one can choose either SVD or MOS depending on the dimension of the local data. To shorten the presentation, we assume the SVD is used for all local data.

Using perturbation theory, we now show that APMOS produces an arbitrarily close approximation to $\mathbf{S}^\intercal\mathbf{S}$.

Theorem 2.1. *Let λ_j be the jth largest eigenvalue of $\mathbf{W}^r(\mathbf{W}^r)^\intercal$ and $\overline{\lambda}_j$ the jth largest eigenvalue of $\mathbf{W}(\mathbf{W})^\intercal$. For $1 \le j \le m$,*

$$|\lambda_j - \overline{\lambda}_j| \le p\,\epsilon_0^2. \tag{9}$$

Table 1
Comparison of computational complexity and communication effort. Terms in curly brackets represent operations to be executed in parallel.

Method	Complexity (flops)	Communication
SVD	$\mathcal{O}(n^2m + nm^2 + m^3)$	nm
MOS	$\mathcal{O}(nm^2 + rnm + m^3)$	nm
PSVD (non-iterative)	$\left\{\mathcal{O}\left(\sum_{i=1}^{p}(n_i^2m + n_im^2 + m^3 + n_irm)\right)\right\} + \mathcal{O}\left(r^3p^3 + n^2r + nr^2 + r^3\right)$	$nr + mr + mpq$
PMOS	$\left\{\mathcal{O}(\sum_{i=1}^{p} n_im^2 + \sum_{i=1}^{p} n_irm)\right\} + \mathcal{O}(m^3)$	$m^2p + mr$
APMOS	$\left\{\mathcal{O}\left(\sum_{i=1}^{p}(n_i^2m + n_im^2 + m^3 + n_irm)\right)\right\} + \mathcal{O}\left((\sum_{i=1}^{p} r_i)^3\right)$	$m\sum_{i=1}^{p} r_i + mr + r$

Proof. Let $A = \mathbf{W}^r(\mathbf{W}^r)^\intercal$ and $\delta K = \mathbf{W}(\mathbf{W})^\intercal - \mathbf{W}^r(\mathbf{W}^r)^\intercal$. Since A and δK are $m \times m$ symmetric matrices, by Corollary 8.1-3 in [12], the distance between individual eigenvalues of A and $A + \delta K$ can be bounded as follows:

$$\lambda_m(\delta K) \leq \lambda_j(A) - \lambda_j(A + \delta K) \leq \lambda_1(\delta K).$$

Then

$$|\lambda_j(A) - \lambda_j(A + \delta K)| \leq \|\delta K\|_2,$$

as the 2-norm of the symmetric matrix δK is equal to its spectral radius, which is the maximum value of $|\lambda_1(\delta K)|$ and $|\lambda_m(\delta K)|$. Since $\overline{\lambda}_j = \lambda_j(A + \delta K)$, we have $|\lambda_j - \overline{\lambda}_j| \leq \|\delta K\|_2$. Based on the construction of $\mathbf{W}^r$, we have

$$\delta K = \sum_{i=1}^{p} \mathbf{V}_i^c(\mathbf{\Sigma}_i^c)^2(\mathbf{V}_i^c)^\intercal,$$

where $\mathbf{V}_i^c = \mathbf{V}_i - \mathbf{V}_i^{r_i}$ and $\mathbf{\Sigma}_i^c = \mathbf{\Sigma}_i - \mathbf{\Sigma}_i^{r_i}$. Since $\|\mathbf{V}_i^c(\mathbf{\Sigma}_i^c)^2(\mathbf{V}_i^c)^\intercal\|_2 \leq \epsilon_0^2$, we have

$$\|\delta K\|_2 \leq p\,\epsilon_0^2,$$

which implies

$$|\lambda_j - \overline{\lambda}_j| \leq p\,\epsilon_0^2.$$

This completes the proof. □

Remark 2.2. Even on serial computers, the proposed partitioned MOS can provide flexibility in treating potential memory issues that may arise when processing large snapshot data.

2.3. Comparison

We now tabulate the computational complexity and communication effort associated with each algorithm we have discussed (Table 1).

Clearly, the partitioned methods of snapshots (Algorithms 2 and 3) are highly parallel, potentially distributing the bulk of their computation tasks over multiple local processors. When the number of snapshots m is large, Algorithm 3 requires a much lower communication effort than the other approaches. In general, choosing between one of these two algorithms should improve the efficiency of producing ROMs for complex dynamical systems modeled on distributed memory parallel computers.

3. Numerical tests

In this section, we illustrate the proposed method in two different test cases: (i) the one-dimensional (1D) Burgers equation; and (ii) a two-dimensional (2D) gravity current model that exhibits complex flow structures. Since Algorithm 2 outputs exact POD basis functions, we only provide examples that demonstrate the performance of Algorithm 3.

3.1. Burgers equation

We first consider the Burgers equation

$$\begin{cases} u_t - \nu\, u_{xx} + u\, u_x = f & \text{in } \Omega \times (0, T], \\ u(x, 0) = u_0(x) & \text{in } \Omega, \\ u(x, t) = 0 & \text{on } \partial\Omega \times (0, T], \end{cases} \tag{10}$$

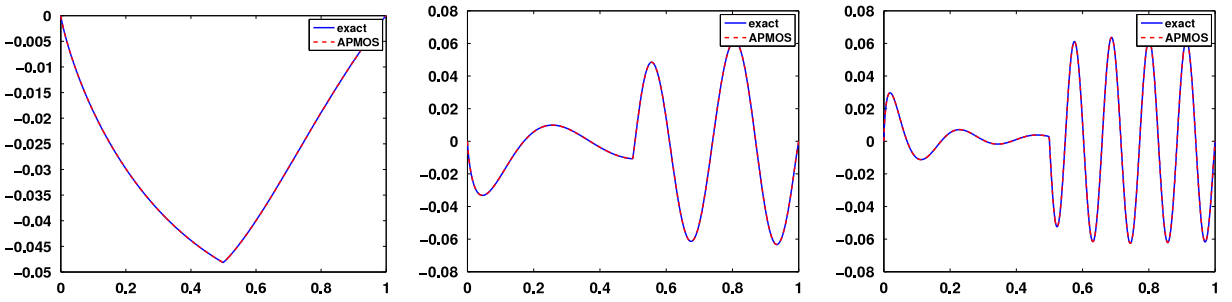

Fig. 1. The first, fifth and tenth POD basis functions for the 1D Burgers equation with $p = 4$. The exact POD basis functions (blue) and the POD basis generated by APMOS (red). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 2
The 1D Burgers equation. Errors in the POD basis functions generated by APMOS with $p = 4$ and different values of tolerance ϵ_0.

Error	$\epsilon_0 = 0.1$	$\epsilon_0 = 0.01$	$\epsilon_0 = 0.001$
$\|\varphi_1 - \widetilde{\varphi}_1\|$	1.4466e−10	5.5463e−13	3.3858e−15
$\|\varphi_2 - \widetilde{\varphi}_2\|$	1.8714e−09	6.6758e−12	7.8371e−14
$\|\varphi_3 - \widetilde{\varphi}_3\|$	1.6335e−08	4.0637e−11	5.0690e−13
$\|\varphi_4 - \widetilde{\varphi}_4\|$	4.5670e−08	3.3525e−10	1.8772e−12
$\|\varphi_5 - \widetilde{\varphi}_5\|$	9.0292e−08	8.5365e−10	6.2636e−12
$\|\varphi_6 - \widetilde{\varphi}_6\|$	2.8252e−07	1.6294e−09	1.0054e−11
$\|\varphi_7 - \widetilde{\varphi}_7\|$	4.5482e−07	2.5468e−09	1.8999e−11
$\|\varphi_8 - \widetilde{\varphi}_8\|$	6.2751e−07	2.6742e−09	1.2333e−11
$\|\varphi_9 - \widetilde{\varphi}_9\|$	1.4773e−06	2.0758e−09	4.3521e−11
$\|\varphi_{10} - \widetilde{\varphi}_{10}\|$	9.5416e−07	8.4396e−09	2.0516e−10

Table 3
The 1D Burgers equation. Maximum errors in the eigenvalues of snapshot covariance matrix generated by APMOS with $p = 4$ and different values of tolerance ϵ_0.

Error	$\epsilon_0 = 0.1$	$\epsilon_0 = 0.01$	$\epsilon_0 = 0.001$
$\max_{j\in[1,10]} \|\lambda_j - \overline{\lambda}_j\|$	3.8796e−04	1.9605e−06	3.1010e−08

where $\Omega = [0, 1]$ and $T = 1$. We specify a small diffusion parameter $\nu = 1 \times 10^{-3}$ and the discontinuous initial condition

$$u_0(x) = \begin{cases} 1 & \text{if } x \in \left(0, \frac{1}{2}\right] \\ 0 & \text{if } x \in \left(\frac{1}{2}, 1\right). \end{cases} \tag{11}$$

The domain decomposition simulation is first performed to obtain snapshots over 4 processors, in which the backward Euler method (time step $\Delta t = 1 \times 10^{-3}$) and linear finite elements (mesh size $h = 1/1024$) are chosen for time and space discretization, respectively. Snapshots are collected at every time step. Thus, the size of the snapshot matrix $\mathbf{S}$ is 1025×1001.

For comparison, we first compute the POD basis from the full snapshot matrix. The first $r_0 = 10$ POD basis functions capture 99.14% of the system's kinetic energy. We then perform Algorithm 3, where we select r_i right singular vectors from the local data such that $\sigma_i^{r_i+1} < \epsilon_0$ on each subdomain, generate $\mathbf{W}^r$ and do SVD on $\mathbf{W}^r$. After that, subdomain-specific components of the POD basis functions on each subdomain are computed respectively. To test the new algorithm, we vary the values of ϵ_0. When $\epsilon_0 = 10^{-1}$, the shape of $\mathbf{W}^r$ is 1001×142; $\epsilon_0 = 10^{-2}$, the shape of $\mathbf{W}^r$ is 1001×203; $\epsilon_0 = 10^{-3}$, the shape of $\mathbf{W}^r$ is 1001×262.

The first, fifth and tenth approximate POD basis functions when $\epsilon_0 = 10^{-1}$ are plotted in Fig. 1 along with the corresponding exact POD basis functions. The L_2-norms of the differences between the first ten basis functions from APMOS and the exact basis functions are listed in Table 2. These comparisons demonstrate that the outcomes of APMOS (Algorithm 3) yield accurate basis function approximations for all ϵ_0 values. As expected, the accuracy increases with decreasing ϵ_0.

The maximum error in the top ten eigenvalues of the snapshot covariance matrices generated by APMOS with $p = 4$ and different ϵ_0 are listed in Table 3. It is seen that $\max_{j\in[1,10]} |\lambda_j - \overline{\lambda}_j|$ is bounded by $p\epsilon_0^2$, which is consistent with the conclusion drawn in Theorem 2.1.

When we vary the number of processors used in the numerical simulations, the difference between the exact POD basis and APMOS results when $\epsilon = 10^{-2}$ are listed in Table 4, which demonstrates that the approximation errors maintain consistent order as the number of processors increases.

Table 4
The 1D Burgers equation. Errors in the POD basis functions generated by APMOS when $\epsilon_0 = 0.01$ and the number of subdomains p varies.

p	2	3	4
$\|\varphi_1 - \widetilde{\varphi}_1\|$	1.6247e−13	4.7123e−13	5.5463e−13
$\|\varphi_2 - \widetilde{\varphi}_2\|$	8.0568e−12	1.2579e−11	6.6758e−12
$\|\varphi_3 - \widetilde{\varphi}_3\|$	6.6596e−11	8.2510e−11	4.0637e−11
$\|\varphi_4 - \widetilde{\varphi}_4\|$	8.9645e−11	3.2307e−10	3.3525e−10
$\|\varphi_5 - \widetilde{\varphi}_5\|$	5.5112e−10	3.3758e−10	8.5365e−10
$\|\varphi_6 - \widetilde{\varphi}_6\|$	9.3517e−10	1.5524e−09	1.6294e−09
$\|\varphi_7 - \widetilde{\varphi}_7\|$	1.8012e−09	1.8414e−09	2.5468e−09
$\|\varphi_8 - \widetilde{\varphi}_8\|$	3.0421e−09	4.2488e−09	2.6742e−09
$\|\varphi_9 - \widetilde{\varphi}_9\|$	4.5354e−09	9.0200e−09	2.0758e−09
$\|\varphi_{10} - \widetilde{\varphi}_{10}\|$	6.6651e−09	1.2987e−09	8.4396e−09

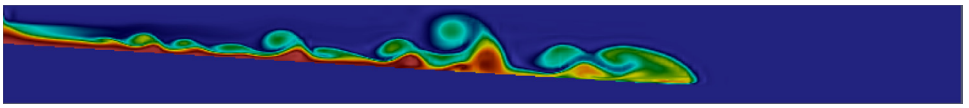

Fig. 2. Gravity current. The temperature snapshot at the final time.

3.2. Gravity current

A gravity current is the flow exhibited in two different fluids driven by the gravitational force acting on the density difference between the fluids. Such a current naturally occurs in the atmosphere and oceans. For example, gravity currents flow down a sloping sea floor in the Strait of Gibraltar and the Denmark Strait. They are key components of thermohaline circulation, which in turn is important to climate and weather studies. Numerical studies of ocean gravity currents based on a nonhydrostatic approximation have been performed in [15]. The mathematical model uses the Boussinesq approximation:

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} - \Delta \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \mathrm{Ra}\, T\mathbf{k} &= 0, \\ \nabla\cdot\mathbf{u} &= 0, \\ \dfrac{\partial T}{\partial t} + (\mathbf{u}\cdot\nabla)T - \mathrm{Pr}^{-1}\Delta T &= 0, \end{cases} \tag{12}$$

where $\mathbf{u} = (u, v, w)$ is the velocity vector, p is the pressure, T is the temperature deviation from a background value, and $\mathbf{k}$ is the unit normal vector in the vertical direction. The Rayleigh number, $\mathrm{Ra} = (g\beta\Delta T H^3)/\nu^2$, measures the ratio of buoyancy and viscous forces, where g is the gravitational acceleration, β is the temperature contraction coefficient, ΔT is the amplitude of the temperature range, and H is the domain depth. $\mathrm{Pr} = \nu/K$ is the Prandtl number, which measures the ratio of viscosity and temperature diffusivity.

In the 2D numerical simulations carried out in [15], $\mathrm{Ra} = 5 \times 10^6$ and $\mathrm{Pr} = 7$. The spatial domain is configured with a horizontal length of $L = 10$ km. The depth of the water column ranges from $K = 400$ m at $x = 0$ to $H = 1000$ m at $x = 10$ km over a constant slope. The slope angle is $\theta = 3.5°$, which is within the general range of oceanic overflows, such as the Red Sea overflow entering the Tadjura Rift. The velocity has a homogeneous Dirichlet boundary condition on the bottom, nonhomogeneous Dirichlet at the inlet, free slip on top, and zero normal flux at the outlet. The model is driven by velocity and temperature forcing profiles at the inlet. The spectral element method is used for the spatial discretization where the mesh contains $n = 14\,400$ grid points and the time step is $\Delta t = 10^{-6}$. The data set contains $m = 900$ snapshots (see Fig. 2).

For comparison purposes, we first compute the POD basis from the full snapshot matrix. The first $r_0 = 10$ POD basis functions capture 88.10% of the kinetic energy in the simulation. We next use the APMOS method (Algorithm 3) on 10 subdomains. We select r_i right singular vectors from the local data such that $\sigma_i^{r_i+1} < \epsilon_0$ on each subdomain, generate $\mathbf{W}^r$ and do an SVD of $\mathbf{W}^r$. As in the first example, components of the first 10 POD basis functions on each subdomain are computed respectively. To test Algorithm 3, we vary the values of ϵ_0. The shapes of $\mathbf{W}^r$ are 900×357 when $\epsilon_0 = 10^{-1}$, 900×389 when $\epsilon_0 = 10^{-2}$, and 900×406 when $\epsilon_0 = 10^{-3}$.

The first, fifth and tenth POD basis functions obtained from Algorithm 3 when $\epsilon_0 = 10^{-1}$ are plotted in Fig. 3. The errors in the POD basis approximated by APMOS are listed in Table 5 as ϵ_0 changes. Again, we observe that APMOS yields accurate basis function approximations for all ϵ_0 values. As expected, the accuracy increases with decreasing ϵ_0.

Maximum eigenvalue errors of the APMOS method when $p = 10$ are listed in Table 6. Again, we see that $\max_{j\in[1,10]} |\lambda_j - \overline{\lambda}_j|$ is bounded by $p\epsilon_0^2$, supporting the conclusion drawn in Theorem 2.1.

Similarly, when we vary the number of processors used in the numerical simulations, the difference between the exact POD basis functions and results of APMOS when $\epsilon = 10^{-2}$ (listed in Table 7) show retention of high accuracy. This time we see that the approximation accuracy degrades somewhat compared to $p = 2$ or even $p = 6$ but still holds an accuracy on the order of 10^{-8} when the number of processors increases to 10.

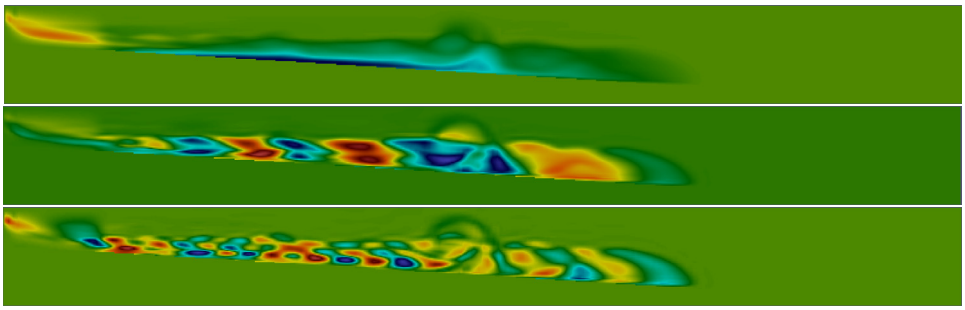

Fig. 3. Gravity current. From the top: the first, fifth and tenth POD basis functions of the temperature generated by APMOS.

Table 5
Gravity current. Errors of the POD basis functions generated by APMOS with $p = 10$ and different values of ϵ_0.

Error	$\epsilon_0 = 0.1$	$\epsilon_0 = 0.01$	$\epsilon_0 = 0.001$
$\|\varphi_1 - \widetilde{\varphi}_1\|$	1.3773e−07	1.4887e−10	8.0819e−13
$\|\varphi_2 - \widetilde{\varphi}_2\|$	3.1577e−07	3.9002e−09	1.3181e−11
$\|\varphi_3 - \widetilde{\varphi}_3\|$	8.4477e−07	1.6343e−08	8.4092e−11
$\|\varphi_4 - \widetilde{\varphi}_4\|$	1.5442e−06	5.1908e−08	2.0944e−10
$\|\varphi_5 - \widetilde{\varphi}_5\|$	1.9823e−06	6.5121e−08	2.7802e−10
$\|\varphi_6 - \widetilde{\varphi}_6\|$	2.3384e−06	4.5175e−08	5.5131e−10
$\|\varphi_7 - \widetilde{\varphi}_7\|$	4.5517e−06	2.9049e−08	2.6552e−09
$\|\varphi_8 - \widetilde{\varphi}_8\|$	2.2137e−05	3.9492e−08	2.9162e−09
$\|\varphi_9 - \widetilde{\varphi}_9\|$	2.3534e−05	4.2363e−08	1.1833e−09
$\|\varphi_{10} - \widetilde{\varphi}_{10}\|$	5.0482e−06	2.5171e−08	3.1499e−10

Table 6
Gravity current. Errors of the eigenvalues of snapshot covariance matrix generated by APMOS with $p = 10$ and different values of ϵ_0.

Error	$\epsilon_0 = 0.1$	$\epsilon_0 = 0.01$	$\epsilon_0 = 0.001$
$\max_{j\in[1,10]} \|\lambda_j - \overline{\lambda}_j\|$	3.9457e−03	2.3582e−05	1.3043e−07

Table 7
Gravity current. Errors of the POD basis functions generated by APMOS when $\epsilon_0 = 0.01$ and the number of subdomains p varies.

p	2	6	10
$\|\varphi_1 - \widetilde{\varphi}_1\|$	1.0371e−13	1.6918e−10	1.4887e−10
$\|\varphi_2 - \widetilde{\varphi}_2\|$	2.0435e−12	2.0478e−09	3.9002e−09
$\|\varphi_3 - \widetilde{\varphi}_3\|$	9.7710e−13	1.0361e−08	1.6343e−08
$\|\varphi_4 - \widetilde{\varphi}_4\|$	1.3924e−11	1.7824e−08	5.1908e−08
$\|\varphi_5 - \widetilde{\varphi}_5\|$	1.4073e−11	2.1721e−08	6.5121e−08
$\|\varphi_6 - \widetilde{\varphi}_6\|$	7.1811e−11	8.6786e−09	4.5175e−08
$\|\varphi_7 - \widetilde{\varphi}_7\|$	2.5553e−10	6.7104e−09	2.9049e−08
$\|\varphi_8 - \widetilde{\varphi}_8\|$	3.8947e−10	2.0783e−08	3.9492e−08
$\|\varphi_9 - \widetilde{\varphi}_9\|$	3.8889e−10	2.2909e−08	4.2363e−08
$\|\varphi_{10} - \widetilde{\varphi}_{10}\|$	2.6685e−10	2.0106e−08	2.5171e−08

4. Conclusions

In summary, we have presented a novel partitioned method of snapshots, APMOS (Algorithm 3), for efficiently generating a POD basis. Compared with PMOS (Algorithm 2), the new algorithm yields an accurate approximation of the POD basis while reducing the communication load. Since it considers distributed features of the snapshot data over multiple processors in a parallel computing environment, the main computational cost of POD basis generation is simultaneously distributed across multiple local processors, which allows it to be executed in parallel for potentially dramatic time savings. The APMOS is certainly very well-suited to simulations typically run in a domain decomposition setting. Furthermore, even on serial computers, this approach can provide flexibility in overcoming potential memory issues caused by large snapshot data. In the future, we will investigate whether the new APMOS method can significantly reduce the CPU time and storage costs [16] of standard POD-ROM numerical discretizations of more realistic complex fluid flows. We note that the DDM has recently been combined with various model reduction techniques, e.g., balanced truncation model reduction [17–19], POD [20–23],

reduced basis method and related variations [24–31]. We will also investigate whether the synthesis of APMOS with POD and DDM can yield further reductions in computational costs of POD-ROMs.

Acknowledgments

The authors thank the anonymous reviewers for their constructive comments, which helped improve the manuscript.

References

[1] S.B. Pope, Turbulent Flows, Cambridge University Press, 2000.
[2] A. Quarteroni, A. Valli, Domain Decomposition Methods for Partial Differential Equations, Oxford University Press, USA, 1999.
[3] A. Toselli, O. Widlund, Domain Decomposition Methods—Algorithms and Theory, Springer Verlag, 2005.
[4] J. Burkardt, M. Gunzburger, C. Webster, Reduced order modeling of some nonlinear stochastic partial differential equations, Int. J. Numer. Anal. Model. 4 (3–4) (2007) 368–391.
[5] K. Kunisch, S. Volkwein, Control of the Burgers equation by a reduced-order approach using proper orthogonal decomposition, J. Optim. Theory Appl. 102 (2) (1999) 345–371.
[6] L. Sirovich, Turbulence and the dynamics of coherent structures. Parts I–III, Quart. Appl. Math. 45 (3) (1987) 561–590.
[7] V. Hernández, J.E. Román, A. Tomás, A robust and efficient parallel SVD solver based on restarted Lanczos bidiagonalization, Electron. Trans. Numer. Anal. 31 (2008) 68–85.
[8] C.H. Bischof, Computing the singular value decomposition on a distributed system of vector processors, Parallel Comput. 11 (2) (1989) 171–186.
[9] C. Beattie, J. Borggaard, S. Gugercin, T. Iliescu, A domain decomposition approach to POD, in: Proceedings of the 45th IEEE Conference on Decision and Control, 2006.
[10] P. Holmes, J.L. Lumley, G. Berkooz, Turbulence, Coherent Structures, Dynamical Systems and Symmetry, Cambridge University Press, Cambridge, UK, 1996.
[11] B. Smith, P. Bjorstad, W. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, 2004.
[12] G.H. Golub, C.F. Van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, MD, USA, 1996.
[13] A. Quarteroni, R. Sacco, F. Saleri, Numerical Mathematics, Vol. 37, Springer Science & Business Media, 2010.
[14] L.N. Trefethen, D. Bau III, Numerical Linear Algebra, SIAM, 1997.
[15] T.M. Özgökmen, P.F. Fischer, J. Duan, T. Iliescu, Three-dimensional turbulent bottom density currents from a high-order nonhydrostatic spectral element model, J. Phys. Oceanogr. 34 (9) (2004) 2006–2026.
[16] D. Amsallem, C. Farhat, Interpolation method for adapting reduced-order models and application to aeroelasticity, AIAA J. 46 (7) (2008) 1803.
[17] K. Sun, R. Glowinski, M. Heinkenschloss, D.C. Sorensen, Domain decomposition and model reduction of systems with local nonlinearities, in: G. Of, O. Steinbach, K. Kunisch (Eds.), Numerical Mathematics and Advanced Applications, Springer, Heidelberg, 2008.
[18] H. Antil, M. Heinkenschloss, R.H.W. Hoppe, D.C. Sorensen, Domain decomposition and model reduction for the numerical solution of PDE constrained optimization problems with localized optimization variables, Comput. Vis. Sci. 13 (6) (2010) 249–264.
[19] H. Antil, M. Heinkenschloss, R.H.W. Hoppe, Domain decomposition and balanced truncation model reduction for shape optimization of the Stokes system, Optim. Methods Softw. 26 (4–5) (2011) 643–669.
[20] P. Kerfriden, O. Goury, T. Rabczuk, S.P.-A. Bordas, A partitioned model order reduction approach to rationalise computational expenses in nonlinear fracture mechanics, Comput. Methods Appl. Mech. Engrg. 256 (2013) 169–188.
[21] J. Baiges, R. Codina, S. Idelsohn, A domain decomposition strategy for reduced order models. Application to the incompressible Navier–Stokes equations, Comput. Methods Appl. Mech. Engrg. 267 (2013) 23–42.
[22] A. Radermacher, S. Reese, Model reduction in elastoplasticity: proper orthogonal decomposition combined with adaptive sub-structuring, Comput. Mech. 54 (3) (2014) 677–687.
[23] Q. Liao, K. Willcox, A domain decomposition approach for uncertainty analysis, SIAM J. Sci. Comput. 37 (1) (2015) A103–A133.
[24] Y. Maday, E.M. Rønquist, A reduced-basis element method, J. Sci. Comput. 17 (1–4) (2002) 447–459.
[25] Y. Maday, E.M. Ronquist, The reduced basis element method: application to a thermal fin problem, SIAM J. Sci. Comput. 26 (1) (2004) 240–258.
[26] A.E. Løvgren, Y. Maday, E.M. Rønquist, A reduced basis element method for the steady Stokes problem, ESAIM Math. Model. Numer. Anal.-Model. Math. Anal. Numer. 40 (3) (2006) 529–552.
[27] A.E. Løvgren, Y. Maday, E.M. Rønquist, The reduced basis element method for fluid flows, in: Analysis and Simulation of Fluid Dynamics, Springer, 2007, pp. 129–154.
[28] Y. Chen, J.S. Hesthaven, Y. Maday, A seamless reduced basis element method for 2D Maxwell's problem: an introduction, in: Spectral and High Order Methods for Partial Differential Equations, Springer, 2011, pp. 141–152.
[29] L. Iapichino, A. Quarteroni, G. Rozza, A reduced basis hybrid method for the coupling of parametrized domains represented by fluidic networks, Comput. Methods Appl. Mech. Engrg. 221 (2012) 63–82.
[30] L. Iapichino, Reduced basis methods for the solution of parametrized PDEs in repetitive and complex networks with application to CFD (Ph.D. thesis) EPFL, 2012.
[31] D.B.P. Huynh, D.J. Knezevic, A.T. Patera, A static condensation reduced basis element method: approximation and a posteriori error estimation, ESAIM Math. Model. Numer. Anal. 47 (01) (2013) 213–251.