Integrating a Reduced-order Model Server into the Engineering Design Process

Sunil Suram and Kenneth M. Bryden*

Department of Mechanical Engineering

Iowa State University, Ames, IA, 50011

*Contact information:

Kenneth M. Bryden
1620 Howe Hall
Iowa State University
Ames, IA 50011

Email: kmbryden@iastate.edu
Ph: 1-515-294-3891
Fax: 1-515-294-3261

© 2015. This manuscript version is made available under the Elsevier user license
<http://www.elsevier.com/open-access/userlicense/1.0/>

Abstract—Engineering design is a complex and iterative process that involves multiple engineering teams sharing and communicating information during the design process. One aspect of engineering design involves the development of physics-based models and their analysis via numerical simulations that are computationally expensive. To overcome the time constraints due to the complexity of numerical simulations, reduced-order models (ROM) such as proper orthogonal decomposition are being increasingly used. Decreasing the simulation time, however, does not address the inefficiencies in communicating engineering models and analysis during the design process. This paper proposes developing and incorporating a ROM server into the design workflow. The ROM server stores all data associated with a given engineering model and automatically constructs a ROM every time a model is created or updated, thus maintaining a consistent version of information across multiple engineering teams. A common engineering workflow is compared with one using a ROM server. A cost of synchronization metric has been defined based on the parameters of data size, size of the engineering team and design iterations. This metric has been evaluated and compared for the cases with and without a ROM server and it was found that the cost of synchronization is lower when a ROM server is used in the design workflow. It is shown that as the team size increases, the ROM server helps with more efficient information storage and transfer. Finally, an example problem of a heat-exchanger fin shape design is used to demonstrate the ROM server framework.

Keywords: *Engineering design; Proper orthogonal decomposition; Client-Server; Distributed computing; Reduced-order modeling*

1. Introduction

Engineering is increasingly dependent on modeling and simulation for design, optimization, and many other engineering decision-making tasks. With increasing computational capabilities it has become easier to run large-scale, physics-based high-fidelity simulations and analyze various physical phenomena. Examples of these simulation techniques include computational fluid dynamics (CFD), finite element analysis, and molecular dynamics. Often this involves discretizing the domain into a finite number of grid points and solving the governing, coupled partial differential equations over the discretized domain. Resolving physical phenomena at these levels of fidelity is time-consuming, and the computational complexity is a function of the number of grid points considered, the complexity of the geometry, and the physics represented by the equation set. As a result, running the numerous “what-if” scenarios needed to support a simulation-based engineering and design process is often computationally prohibitive.

To overcome this limitation, various data-driven reduced-order modeling techniques have been developed. Data-driven reduced-order modeling relies on first creating a collection of computational solutions to construct a set of basis functions. These basis functions are then used to make evaluations of the reduced-order model (ROM) in lieu of using the original large-scale physics-based computational model [5,8,9,13]. One reduced-order modeling technique is proper orthogonal decomposition (POD). POD-based ROMs have been used in a number of engineering and scientific applications, including fluid mechanics [3,8,11,13] and solid mechanics [25]. For example, Suram et al. constructed a ROM based on CFD simulations of flow through a mixing nozzle [3]. Samadiani et al. reviewed design methodologies for data-center server thermal management based on the POD method [5]. Willcox et al. developed an inverse design technique based on a POD technique for incomplete data [9]. Zhou et al. used POD to analyze turbulent flow structures in a reacting jet [25]. Du et al. have incorporated the POD technique into a finite difference scheme and have analyzed the errors after applying this unified scheme to develop a ROM for a chemical vapor deposition reactor [29]. The generality of the POD methodology also makes it useful in any field that involves studying and analyzing patterns of data that have been collected either computationally or experimentally. One interesting example where POD-based ROMs have been used with experimental data is in Chen et al. where the authors have analyzed bat flight kinematics from data collected from video samples [24]. In engineering design, the focus of POD is to reduce the time to run the computational models within the engineering workflow. That is, the high fidelity models are created and are then used to create the ROM. The ROM is then used in the design process.

Within this process it is often assumed that the ROM creation process is a linear and static process and that design exploration, optimization, and decision-making wait for the creation of the ROM. And that once the ROM is created, it does not change. However, engineering design is a dynamic process of exploration and consideration of multiple design options within a broad analysis space. Because of this, detailed high fidelity computational modeling is often delayed until the design is nearing completion. Pushing this detailed modeling forward in the design process has the potential to reduce costs and improve designs, but it requires a framework in which the existing computational results can be utilized and updated while the analysis process proceeds. That is, the analysts and engineers need a common framework that enables a shared design-analysis workflow.

To meet this need, this paper proposes a client-server based architecture to build and evaluate POD-based ROMs in which the inputs from multiple engineers and analysts are incorporated and managed as a part of a dynamic and shifting engineering design process. This ROM server enables independent insights obtained by the designers and analysts within each iteration of a design to be used to improve successive iterations. In addition, integrating the various steps into a cohesive workflow enables faster, more consistent, and more predictable information sharing within the engineering design team, which may result in shorter more effective design cycles.

2. Background

Engineering design is an iterative decision making process in which collaborative groups of designers and engineers work together from the conceptual design to a final product. Many engineering design workflows have been proposed but most of these are similar to Fig. 1 [30,31]. As shown in Fig. 1, the design process is composed of three main stages; (1) problem definition, (2) engineering design, and (3) design validation and verification. The engineering design stage can be thought of as consisting of three phases: conceptual design, preliminary design, and detailed design [30, 31]. During conceptual design engineers explore the design space through the generation of concepts that then are filtered using the constraints defined for the problem. Following conceptual design, preliminary design further refines these concepts to one design. During the detailed design phase the chosen design is optimized and finalized. High fidelity modeling offers the power to improve and support creative engineering design in the exploration of ideas, which occurs during the conceptual design and preliminary design phases. But because of the time and expense required to develop, execute, and process these high fidelity models, they are typically used primarily during the detailed design phase. In contrast to the task oriented approach to the development of high fidelity models, engineering design is an iterative process in which the designers typically

move back-and-forth between the conceptual, preliminary, and detailed design phases with relatively quick consideration and analysis of multiple designs, repeated iterations and expansions of proposed solutions, revisiting assumptions and decisions, and a series of design changes. Once completed a reduced set of designs are chosen for further verification and validation using additional analysis and field data. This can lead to changes to the initial design, thus requiring a repeat of the design stage. The exploratory and iterative nature of engineering design makes the process of communicating engineering information and analysis during the design stage between various engineering teams challenging.

2.1. Proper Orthogonal Decomposition

This section provides a brief discussion of the POD technique, which is needed to understand the implementation of the ROM server. Readers desiring a more detailed discussion of the POD technique are referred to [2]. The POD technique is used to find a set of optimal truncated orthogonal basis functions from a set of snapshot solutions. These snapshot solutions are solutions that span the space of interest. In engineering design these are typically from numerical simulations of the system or phenomena of interest. Within the space defined by the set of snapshot solutions, a solution vector, $\bar{x}_{sol}$, can be found using a set of basis functions, u^i .

$$\bar{x}_{sol} = \sum_{i=1}^D a_i u^i \quad (1)$$

where D is the dimension of the truncated vector space and the a_i are the coefficients that are used to compute the POD approximation for a given set of basis functions.

To find the optimal set of truncated basis functions needed for the ROM, the first step is to generate a dataset of M snapshot solutions that span the engineering design space of interest. The snapshot solution dataset is then centered by computing and subtracting the mean of the dataset from each snapshot. The mean-subtracted M snapshots are then concatenated in a single ensemble matrix, $\mathbf{X}_{N \times M}$, where N is the size of each snapshot and M is the number of snapshots. Once the ensemble matrix is assembled, the POD basis functions are computed from the covariance of the ensemble matrix using singular value decomposition. The coefficients are then found by projecting the POD basis functions onto the original ensemble matrix.

For a given ensemble matrix the basis functions are constant, and the coefficients are associated with the design space that was explored by the dataset of snapshot solutions used to create the ensemble matrix. To evaluate a design for a set of parameters that are not a part of the dataset of snapshot solutions but are within the initial design

space covered by the analysis, linear interpolation is performed on the coefficients as shown in Eq. (2). For example, if the coefficient vector, $\vec{a}^*$ at a given design parameter vector, $\vec{q}^*$ such that $\vec{q}_k < \vec{q}^* < \vec{q}_{k+1}$ have to be evaluated, $\vec{a}^*$ is given by Eq. (2). The POD approximation is then computed using the interpolated coefficients as described in Eq. (1).

$$\vec{a}^* = \vec{a}|_{q_k} + \left(\vec{a}|_{q_{k+1}} - \vec{a}|_{q_k} \right) \frac{(\vec{q}^* - \vec{q}_k)}{(\vec{q}_{k+1} - \vec{q}_k)} \quad (2)$$

The accuracy of the ROM is dependent on the number of terms used in the POD expansion and is determined by Eq. (3). E_i is the “energy” of the POD expansion and s_i represents the i^{th} singular value. It is called the energy because the singular values are equal to the square of the corresponding eigenvalues of the covariance matrix [2].

$$E_i = \frac{s_i}{\sum s_i} \quad (3)$$

Thus the singular values of the covariance matrix $\mathbf{X}_{\text{cov}}$ can be used to determine if sufficient data is included in the initial snapshot set and to determine the accuracy of the ROM. For a design engineer evaluating a ROM, the singular value spectrum can serve as a useful guide to determine if there is sufficient information to make an engineering decision before proceeding to the next step in the design process. If the total energy captured by the dominant singular values is within the acceptable range of error, it can be concluded that the ensemble matrix has captured sufficient information. If this is not the case the ensemble matrix needs to be expanded with more information. The error of the ROM is a function of the number of models chosen to construct the ensemble matrix that is dependent on the energy captured by the singular value spectrum [2].

The POD technique is useful because it captures all the required information about the phase space of a given physical problem. When using it to solve an engineering design problem, this information can then be used in conjunction with the coefficient interpolation technique to explore the design space in a computationally efficient manner. This process can be summarized as follows:

- Identify the design parameters and design space of interest
- Create the computational data needed for the snapshot dataset that spans the design space of interest
- Create the POD coefficient and basis functions
- Make the POD ROM available for use

- Use the POD ROM to compute new solutions as needed to support the engineering design process

If the design space to be explored needs to be expanded or new aspects of the problem need to be explored, the snapshot dataset will need to be expanded and a new POD ROM will need to be developed. Additionally, the accuracy of the POD ROM increases as the number of snapshot solutions increases. Thus as the design process evolves and more accurate solutions are needed, the POD ROM will likely need to be updated in regions of the design space of particular interest.

The iterative nature of the design process and the continuing update of the ROM creates a communication challenge within the design and analysis team. To evaluate a ROM, the most recent set of coefficients and the basis functions need to be known by the user. If a user is geographically in a different region or a part of a different engineering team interested in evaluating the ROMs or analyzing the results, this information has to be made available to them. Updating the ROM manually via email or download for local compute makes it challenging to ensure that the most recent ROM is used and that disparate members of the design group are using the same ROM. Furthermore, providing local access to this data for multiple users may not be possible. It is also likely that multiple POD ROM models would be used to address a large-scale complicated design problem, and a process is needed to coordinate the development and use of these multiple POD ROM models. This creates problems with management of data and version control. The ROM may remain on a single computational machine or may be exported to remote machines for simultaneous use. If it is kept on a single machine, access is limited because only one ROM computation can be performed at a time. If it is exported, maintaining version control of the ROM becomes difficult and different groups having conflicting or out-of-date information can slow the design process. In the next few sections we propose an engineering workflow to overcome these challenges and enable the seamless utilization of ROMs within the engineering design process.

2.2. Reduced-order modeling within a traditional engineering design workflow

In a typical design workflow the design engineers identify the parameters of interest within the proposed design space. The analysts then computationally model the physical phenomena, verify and validate the models, and run the computer simulations needed. If ROMs are being used, the set of computer simulations is used to create the ROMs. The ROMs are then used to explore design space and optimize the design. Overall the creation of the model and the ROM is linear and unidirectional. This contrasts with the dynamic, exploratory nature of the design process. The issues posed by working in a traditional batch compute paradigm to create the ROM include

- To work independently, the consumers of information have to make their own copies of the data. This imposes an inherent bottleneck in the collaborative process.
- If a producer of information makes changes to a computational model or design parameter, the consumers must explicitly synchronize their copies of the data to get the updated information.
- A part of this manual synchronization also involves the manual regeneration of the ROM.
- The creation of the snapshot solutions and the design proceed in series rather than in parallel, and hence analysis process and the design engineering process cannot inform each other of critical decisions and information in real time. Rather, the design waits for the analysis and the analysis waits for the design.

Thus, each of the activities involved are decoupled from one another, resulting in a discontinuous workflow and complicating the task of managing and effectively utilizing computational data.

One solution to the coordination and communication problem posed by the incorporation of high fidelity analysis in the design process is to develop a client-server based engineering design workflow. A client-server architecture has one central computer node acting in the role of a server. Clients connect to this server to request information or use the server's hardware resources to perform a computational task. Once this task is performed information is returned back to the client, completing the transaction. This is a centralized architecture, which implies that the global state needs to be maintained only on one node. This makes the maintenance of a single server relatively straightforward. Because the client typically does not perform the heavy computations, this architecture has the advantage that small form-factor clients (e.g., tablets and other mobile devices) can be developed to perform specific design engineering tasks. Based on this, this paper presents a client-server based architecture to support the integration of high-fidelity modeling using POD ROM modeling into the engineering design workflow.

3. Proposed Engineering Workflow

As noted earlier, the engineering design process is a dynamic, information-rich process that brings together many disciplines to create a product or solution that addresses a given set of needs within a complex and constrained design space. Within this space computational models and information are created and used by varying groups at varying times. Within this simulation-based engineering design environment a workflow that simplifies the creation, use, and update of the ROMs and enables various engineering groups to share this information easily is needed. Fig. 2 shows the proposed engineering design workflow in which a ROM is included as a natural part of the design process. As shown, the ROM server is central to all the interactions at various stages of the engineering design

process. The computational models created and updated and the solutions explored during the design process are all stored in the ROM server. The solution sets generated are utilized to build the reduced-order models that the ROM server publishes for use by various teams. During the design and analysis stages, changes to design parameters, computational constraints, changes to geometry, and other changes can all be stored in the ROM server, which periodically reconciles this data to generate an updated lower-dimensional approximation. Various engineering groups can then use this database of computational models during the different stages of the design process to perform engineering tasks. For example, a CFD analyst can analyze a flow field and a design engineer can work on optimizing shape while yet another engineer can extend the design space by adding more computational models to the database, all doing so simultaneously. In the same way, a design engineer can request that the ROM be extended or improved in a specific way. These interactions are marshalled by the ROM server in such a way that the producers and consumers of computational data have a consistent view of the current state and the requests for new or improved information, and they have access to the same data.

Several researchers have referred to the collection of computational results that are utilized to construct the POD basis functions as a “database” [3,8,11,13,21,22]. Although in the strict sense of the word this collection of computational results is a database, from the perspective of database systems it is missing several key components. It is not accessible to multiple users simultaneously and requires local access to all the data to perform evaluations. In addition, a database system addresses the issues of accessing the database, performing computations, and running optimization problems remotely across a computer network. In the cases cited here, the POD-based ROM evaluation process is a batch process (i.e., the computer program reads the necessary input data, computes the POD approximation for the given input, writes the output to a file and then exits). When the POD approximation needs to be computed for a different set of design parameters, the computer program is restarted. The same also applies to cases where computational datasets are added to or deleted from the ROM database. In these cases the entire database has to be recreated to account for these changes and the POD basis functions have to be recomputed. Each user must then manually update their work, and as the number of users increases the cost of synchronization also increases. In a collaborative team environment this manual synchronization of data and information is a slow and inefficient process. Furthermore, because the run-time of a ROM is much quicker than that of a high-fidelity computational model, running ROMs as a batch process does not allow for efficient sharing of computational information and the fast computations associated with them.

Because a data-driven ROM relies on a computational database, it is an excellent candidate to be treated as a database of engineering data that users can access remotely via a computer network. One way to achieve this is through the use of server-based architecture that manages the ROM process as a part of the engineering design process. Central to this proposed architecture is the ROM server. As shown in Fig. 2, the ROM server provides access to the ROM to all the producers and consumers of the engineering models. The ROM server is accessible over a computer network so ROM computations and analysis can be performed independently by all the consumers simultaneously without involving data copying and manual synchronization. The key activities the ROM server needs to provide are

- *Management and storage of the data*—A group of analysts can be involved in generating the computational snapshots required, based on the needs of the design engineers. This information is used as input to the ROM server, which manages all the computational data and ensures that it is available to other consumers as needed. This requires a mechanism that enables long-term persistence and storage of the data which enables users to access it when required. When an analyst updates this database, the ROM server needs to seamlessly trigger the computational processes that recompute and update the necessary ROM parameters. The updated ROM parameters can then be made available to the engineers, analysts, and decision-makers involved in the engineering process.
- *Managing the ROM creation and update process*—After the snapshots for the ensemble matrix are computed the ROM server needs to collate the snapshots and compute the basis functions and coefficients for the reduced-order model. At this point the server should be ready and listening for client requests to perform ROM evaluations. When a producer updates the ROM database, the data is updated on the persistent storage immediately. However, the recompute of the ROM parameters must be carried out when no users are utilizing the ROM server resources. This is to ensure availability of the service during peak times when multiple engineers might be utilizing the ROM data for analysis.
- *Enabling interactive use and exploration using the ROM*—With data management, storage and the ROM creation processes in place, the constructed ROMs need to be usable by multiple engineers within the organization. Thus, being able to perform ROM evaluations based on user requested input is a critical component of the ROM server. In addition to computing the ROM for a given set of input parameters, the ROM server can also evaluate requests for the singular value spectrum, principal axes, and the projections

of field variables on the dominant principal axes where each of these can be part of the analysis of the physical phenomenon being studied. Another integral piece of design space exploration is the ability to perform engineering optimization utilizing the ROM, which the ROM server must be capable of supporting with an appropriate client that incorporates the optimization algorithm. Because all the data from the aforementioned analyses can be visualized, the ROM server can have the capability to create the outputs needed to support the design process.

- *Maintaining version control*—Many engineering organizations are geographically dispersed with engineers working across multiple time zones. Thus it is important that geographically distributed engineering groups have up-to-date access to the engineering data. When the ROM database is updated, the users automatically receive the updated model parameters from the ROM server, thus providing a unified version of all computational data across all the users in an organization. This ensures that once the model parameters are updated, all users have the exact same information. The ROM server is capable of synchronizing this information seamlessly, which eliminates the cost of explicit data synchronization between the users.

In the proposed engineering workflow the analysts and design engineers are better integrated into the overall engineering design process and can exchange information with greater ease. Real-time synchronicity enables the production and consumption of information independently and helps breakdown the sequential and unidirectional flow of information associated with batch processing.

3.1. Synchronization of data

Section 2 introduced the notion of producers and consumers of data and information in the context of the engineering design process. When a subset of the data changes due to an update from an information producer, other producers as well as consumers need to synchronize their copies of data to ensure that their engineering decisions are made based on the latest information. Thus when information needs to be exchanged between multiple producers and consumers, a cost can be computed based on the information to be exchanged so that all the engineers have access to up-to-date information; this is the *cost of synchronization*, α . The cost of synchronization includes the costs of data exchange, computation, and the associated user costs. In this context, the data exchange cost refers to the time that is needed to move data from one computer to another (i.e., a larger volume of data leads to higher data exchange costs). Computational cost is the time taken to perform a computation on a specified dataset. It is a function of the type of computation and size of data. User cost is the communication time between users of

engineering data for either requesting or notifying other users about an update to the data. Because this is an asynchronous form of communication between users, it includes the lag between the intent and the time the action is actually performed. The asynchronous forms of communication include but are not limited to email, phone conversations etc. In distributed engineering teams the user synchronization cost can be high and difficult to estimate. In this section several cases are outlined that determine the cost of synchronization between multiple producers and consumers considering various amounts of information exchanged when a ROM server is not used. Comparisons are then made to the corresponding cases when a ROM server is used to emphasize the decrease in user costs due to the utilization of a central data repository for engineering data and computations.

The types of data that need to be synchronized between a consumer and a producer using a POD-based ROM can be the entire ensemble matrix, the coefficients in conjunction with the basis functions, and the covariance matrix, each of which is essential for either the producer or the consumer to have the ability to evaluate an engineering model or make an engineering decision. Each has a different cost of synchronization and can be expected to follow the following inequalities

$$\begin{aligned}\alpha_{\mathbf{X}_{\text{cov}}} &< \alpha_{\mathbf{X}} \\ \alpha_{\mathbf{A}} &< \alpha_{\mathbf{U}}\end{aligned}\tag{4}$$

where $\mathbf{X}$ is the original ensemble matrix, $\mathbf{X}_{\text{cov}}$ is the covariance matrix, $\mathbf{A}$ is the coefficient matrix and $\mathbf{U}$ represents the basis functions. By synchronizing the covariance and coefficient matrices, $\mathbf{X}_{\text{cov}}$ and $\mathbf{A}$, respectively, the user can develop ROMs but can only access the initial snapshot models at a lower accuracy. On the other hand, having the entire ensemble matrix $\mathbf{X}$, which has a higher cost of synchronization, one can recreate the basis functions $\mathbf{U}$ and coefficients $\mathbf{A}$ in addition to accessing the initial snapshot models at the same accuracy that they were created at. Thus Eq. (4) signifies the trade-offs that can be made in order to balance the data exchange cost and accuracy of the ROM.

Furthermore, there can be multiple design iterations among the producers and consumers adding to the data synchronization cost because it requires that the information exchange be repeated among the producers and consumers. Thus, enabling synchronicity in design enables engineers to exchange information easily, makes design cycles shorter, and boosts the quality of designs by allowing deeper design space exploration. The remainder of this section examines the cost of data synchronization for each of these scenarios.

3.2. Producer-consumer synchronization

To develop the concept of synchronization cost further, we consider the case of a single producer and a single consumer, as shown in Fig. 3a. The producer has the ability to modify computational models and generate the ROM, whereas the consumer utilizes the ROM to perform further analysis and model evaluation. The consumer can either (a) get the entire ensemble matrix $\mathbf{X}$ from the producer, (b) get a copy of the covariance matrix $\mathbf{X}_{\text{cov}}$ from the producer, or (c) get only the necessary coefficients $\mathbf{A}$ and basis functions $\mathbf{U}$.

- a) In the case where the consumer receives a copy of the ensemble matrix, the coefficients and basis functions can be computed by the consumer and then used for ROM evaluations as needed. The cost of synchronization, α , is thus the time to synchronize the ensemble matrix between producer P_I and consumer C_I and the time to compute the ROM.

$$\alpha_{\text{no-server}} = T_{\mathbf{X}}^{P_I C_I} + t_{\text{ROM}}^{C_I} \quad (5)$$

where T is the communication time and t is the consumer compute time.

- b) Alternatively, because the size of the ensemble matrix $\mathbf{X}$ can be large, $N \times M$, the consumer can get a copy of the covariance matrix $\mathbf{X}_{\text{cov}}$ of size $M \times M$ and compute the ROM parameters. In this case

$$\alpha_{\text{no-server}} = T_{\mathbf{X}_{\text{cov}}}^{P_I C_I} + t_{\text{ROM}}^{C_I} \quad (6)$$

The trade-off in this case is that the consumer does not have all the information about the computational model but has access to sufficient information to compute the ROM and examine the design space

- c) In this case the consumer requests only the ROM coefficients and basis functions, thus making the cost of synchronization, $\alpha_{\text{no-server}}$, a function of synchronizing the $\mathbf{U}$ and $\mathbf{A}$ matrices.

$$\alpha_{\text{no-server}} = T_{\mathbf{U}}^{P_I C_I} + t_{\mathbf{A}}^{P_I C_I} \quad (7)$$

Each of the above cases has inherent advantages and disadvantages in terms of the amount of data that needs to be synchronized as well as any additional computations. It should also be noted that in each of these cases as the number of design iterations increase, the synchronization factor increases linearly by a corresponding factor.

3.3. Multiple producers and consumers

Synchronizing all information is more complicated when there are multiple producers $\{P_1, P_2, \dots, P_K\}$ and multiple consumers $\{C_1, C_2, \dots, C_L\}$ involved in the design process. This scenario is shown in Fig. 3b. When there

are K producers and each of the producers can make changes, there can be K concurrent changes to the ensemble matrix. Consider the scenario where L consumers have to get updates from one or more of the K producers. This is a two-step process which first requires a synchronization among all producers before synchronizing all consumers. For an update to be consistent with the global updates among the producers, all the producers must first synchronize their data. If $\mathbf{X}_c$ is the change to the ensemble matrix by any single producer, the total cost of synchronization for all the producers considering K changes. This is given by

$${}^K P_2 T_{\mathbf{X}_c}^{P_i - P_j}$$

where

$${}^K P_2 = \frac{K!}{(K-2)!}$$

Once this is done the ROM must be recomputed by each producer before a consumer can request an update from any one of the producers. Thus the cost of synchronizing all consumers, α is

$$\alpha_{PC} = ({}^K P_2 T_{\mathbf{X}_c}^{P_i - P_j} + \sum_{i=1}^L T_{U+A}^{P_i C_i} + \sum_{j=1}^K T_{SVD}^{P_j}) \times R \quad (8)$$

where R is the number of design iterations. Eq. (8) shows the high cost of explicit synchronization when there are multiple producers and consumers. The synchronization can be performed implicitly using the ROM server, which reduces the cost of explicit user synchronization and is explained in detail in the following section.

3.4. Synchronization with a ROM server

In this section we study the cost of synchronization when a ROM server is utilized, as shown in Fig. 3c. A major advantage of utilizing a server to synchronize and manage computational data is that producers do not have to perform explicit synzhronization. All data synchronization tasks are performed by the ROM server. Individual producers only have to notify the ROM server of updates that were performed. The ROM server then reconciles the data and manages updated versions of the ROM ensemble matrix. When these are updated, the ROM server also periodically computes the ROM basis functions and coefficients, thus maintaining a unified version of the computational data when accessed by clients.

The total cost of synchronization between any producer P_i and the ROM server R is

$$\alpha_p = \sum_{i=1}^K T_{\mathbf{x}}^{P_i R} \quad (9)$$

where $\mathbf{X}$ denotes a single update from a producer. Similarly the cost α_C for a consumer C_j to get data from the server is given by

$$\alpha_C = \sum_{j=1}^L T_{\mathbf{X}}^{RC_j} \quad (10)$$

Furthermore because the server recomputed the POD approximation periodically for updates to the ROM database, the total computational cost is given by

$$\alpha_{\text{POD}} = \sum t_{\text{SVD}}^R \quad (11)$$

Thus the total cost of synchronization is

$$\alpha_{\text{server}} = \alpha_P + \alpha_C + \alpha_{\text{POD}} \quad (12)$$

Comparing the equations for the synchronization cost in the case of multiple producers and consumers and utilizing the ROM server shows that

$$\alpha_{\text{server}} \ll \alpha_{PC} \quad (13)$$

This result supports the premise that the ROM server reduces the cost of synchronization between producers and consumers by managing the changes to computational data as well as performing the POD computations. Thus, repetitive computations are avoided and connectivity between engineers and analysts in the engineering team is enhanced.

In summary, when a ROM server is not used, there is a significant user cost associated with information synchronization in the engineering design process, whereas using the ROM server eliminates the need for explicit synchronization and hence reduces the user costs. A server-based solution has significant advantages when the engineering team is large and distributed globally. Producers and consumers of engineering models can work and exchange information simultaneously and in a planned manner. Furthermore, having a single repository for all the engineering models in an organization enables users to access the computations and analysis as needed.

4. Design of the ROM server

Typically in scientific computing, when performing tasks over a network or utilizing multiple nodes on a shared memory machine, the emphasis has been on using programming tools like MPI and OpenMP to parallelize code and decrease the runtime. An MPI-based application typically runs on a cluster of computers distributing tasks over them and finally gathering the results. OpenMP on the other hand is designed for shared memory architectures and best

suited for single input multiple data (SIMD) problems [33]. MPI- and OpenMP-based computational solutions do not allow the results to be shared by several users simultaneously across a computer network. Moreover, these compute jobs are typically batch processes (i.e., they run once and have to be rerun for the next numerical experiment). In contrast, the server-based solution needed here involves a server process running continuously in the background and is always available to respond to client requests for ROM evaluations as well as edits to the ROM database. This enables multiple engineers to simultaneously leverage and share the same data while performing independent tasks and computations. For example, one design engineer could be analyzing a flow-field while another engineer simultaneously works on an optimization problem based on the same data, both without local access to the entire ROM database. Furthermore, the server-based ROM database solution helps provide vendor agnostic access to data, computations, and analysis. Specifically, although there are a number of commercial packages available to build engineering models, the ROM server as implemented here has the ability to read in data from most commercial packages utilizing the VTK data format [12]. This gives the server the ability to process and compute ROMs from multiple data sources. In addition, users when accessing the computations and analysis via the ROM server need not have access to the commercial package that was used to generate the initial set of models, thus helping provide vendor agnostic access to engineering models and analysis to experts and non-experts alike.

Thus to get these benefits, the underlying architecture for the ROM server is client-server based. The main purpose of the ROM server is to be available to respond to user requests and perform the required computations in a timely fashion. Because multiple users can be requesting evaluations simultaneously, the server must have a thread pool capable of processing such requests. There are four key building blocks needed:

- Create and manage a computational database that is accessible over a computer network (TCP/IP)
- Create the ROM
- Enable a client node to submit an evaluation request that executes on the ROM server and sends the results back to the client.
- Enable the ROM server to detect additions and deletions of computational data to the database and automatically schedule recomputations of the POD basis functions.

Fig. 4 shows the main components of the ROM server architecture. The dotted lines indicate loading of the POD databases from disk into memory on startup and the solid lines indicate operations of the server after startup. This must be achieved in a manner such that each user gets the correct information back regardless of the number of

concurrent users. If there are edits to the ROM database, these should be reflected in the computations in a timely manner, so that stale information is not used to make engineering decisions on critical components. To achieve fast computations, the server on startup loads the coefficient $\mathbf{A}$ matrix and basis function $\mathbf{U}$ matrix into memory. From then on, as long as the server is running, all POD evaluations are done using the information stored in memory using these matrices and Eq. (1). The design considerations of the ROM server are

- A schema has to be developed to store the computational models and design parameters associated with them on disk.
- For the client and the server to communicate, they both need to be able to interpret data passed from one to another correctly. This requires the construction of communication protocols for the messages that need to be passed between them.
- The server must be capable of accepting client connections and making ROM evaluations as per their requests. This requires the selection of an appropriate data transport layer, in this case TCP/IP. The server must also be capable of accepting multiple client connections from analysts and design engineers, and processing them appropriately. This requires a pool of threads that can coordinate with one another. Because the threads in the thread pool share the same process address space, multiple threads can perform the same operation simultaneously on shared data structures which leads to race conditions [33]. Hence adequate precautions should be taken to enable data sharing while avoiding race conditions.
- The need for the server to recompute the POD basis functions automatically requires a background thread that cannot be invoked by a client. For this purpose a background thread has to be created, and a suitable algorithm has to be developed to enable time-triggered recomputations.

In this work TCP/IP is used as the transport medium for communications between the client and the server [15]. The primary reason is that TCP/IP guarantees ordered delivery of packets, which is critical in this application. Remote procedure calls are used to establish communications between the client and server at the application layer of the networking stack [15]. It must be noted that in this case the availability of the ROM server is on a best-effort basis, and when recomputing the POD basis functions, it is briefly unavailable for client requests. This design trade-off is explained in detail in Section 4.1.2.

4.1. Server thread pool

For a server application it is important that it have the ability to handle multiple client requests simultaneously. In the ROM server, this requirement is handled by having a thread pool containing multiple threads of execution. When a request for information or performing a computation comes in from a client, a free thread from the thread pool is assigned the task, performs the requested task, and sends the information back to the client. Once it is done with this task, the thread marks itself available and waits until another client request is assigned to it. In the example developed here, the ROM server has a total of 10 threads to process client requests in addition to the main thread of execution. The number of threads in the pool is configurable. It was chosen by trying to balance the requirement for multiple client requests and the total number of processors available on the machine, in this case 16. If more processors are available the thread pool size can be increased as needed.

4.2. ROM recomputations

During the engineering design process situations can arise where the design parameter space increases, requiring that additional computational models be added to an existing database. Similarly some design parameters can be deemed unnecessary, which can lead to discarding existing computational models. These scenarios have to be accounted for with minimal disruption to the continual operation of the ROM server to make it an effective tool for managing computational data. Thus, while the ROM server is running, edits should be allowed to the database of computational models. These edits can be datasets that have been added or deleted from the ROM database.

Such edits require recomputation of the POD basis functions and coefficients to update POD evaluations to reflect the new information. The process of updating the POD basis functions leads to multiple software design choices to schedule the recompute task, each of which are briefly explained:

- *Immediate scheduling* in which recomputation is scheduled based on the availability of new information. Immediate scheduling has the inherent advantage of making new information available immediately; however, there exists the possibility that there may be multiple consecutive changes to the ROM database in which the server performs repeated POD basis function recomputations, each superseding the previous one. This creates unnecessary delays in the work and results in unnecessary compute cycles.
- *Delayed scheduling* in which recomputation waits for periods of no client activity to incorporate new data in the ROM database. Delayed scheduling has the advantage that existing requests for computations are given high priority. The primary disadvantage of delayed scheduling is that during periods of prolonged

activity all client requests get stale information, and it requires manual coordination among the clients to cease new requests in order to trigger a POD basis function update.

- *Periodic scheduling* in which recomputation of the POD basis functions is scheduled at a regular time interval that can be configured as per the requirements of the engineering team. The major advantage of periodic scheduling is that multiple changes to the ROM database can be reconciled and computed at once without utilizing compute cycles for each update. This also establishes a deterministic schedule and ensures availability of the same information to the entire engineering team at all times. Thus the producers can work to have the updates done before the recompute, and the consumers know when to expect the next update. The tradeoff in this method is that updates to the ROM database may not be immediately available for use. In the example that follows the authors have chosen this scheduling technique due to its predictable nature and its advantage of being configurable to the work habits of the engineering team.

To accomplish the task of computing the POD basis functions and coefficients while the ROM server is running, a background recomputation thread has been incorporated in the design of the server. This thread is a lazy asynchronous timer thread that checks the ROM database for changes at a specified frequency. Only when an edit is detected does this thread proceed with the recomputation. This is a background thread and does not belong to the thread pool discussed in the previous section, which processes client requests. This thread is in the idle state for most of the time and is alerted by an operating system signal at the end of the specified time period. Once it is out of the idle state, it proceeds to check the timestamp of the last ROM database update and compares it with its own copy from the previous time it was alerted. If a ROM database update has occurred since the previous run, it proceeds to perform a recomputation. Because recomputation of the POD basis functions is a computationally expensive task (i.e., it can take several minutes depending on the size of the ensemble matrix), this thread should be scheduled to run during periods of no utilization of the ROM server.

Furthermore once the POD basis functions are computed, client requests for POD evaluations are not processed for the brief period while the data structures are being replaced in memory. Although it is possible to serve existing client requests with the older version of the ROM, we chose not to do so to ensure that all clients get access to the same information once a recomputation is complete. The time-based triggering algorithm is described in pseudo code as follows:

Algorithm (1)

Parameters : Database update time (tdb), Previous run time of thread (tp)

Procedure (Re-computation)

Step 1 : If (tdb > tp)

 Re-compute POD

 Close all existing client connections

 Stop listening to newer client requests

 Update in-memory data structures

 Re-open server to process client requests

 Sleep

 Else

 Sleep until next alert

 Goto Step 1

End Procedure

5. Application to heat exchanger fin shape design

Section 4 described the design of the ROM server; this section provides an example in which multiple engineers exchange computational models to perform a collaborative engineering task using a ROM server. The various interactions are studied in the absence of and then in the presence of a ROM server. The scenario involves two data producers (P_1 and P_2) and three data consumers (C_1 , C_2 , and C_3). The producers in this case study are responsible for generating computational models, and the consumers perform engineering tasks related to analysis and optimization. Yet another important requirement is that the producers as well as the consumers should have access to all the models as well as the analysis that has been performed by the team as a whole. It should be noted that the focus of this work is on studying the interactions between the producers and consumers and not on the actual analysis of the problem under consideration. A two-dimensional heat exchanger fin shape design problem has been chosen as the engineering design problem in which multiple engineers work on different aspects of the overall design. The designs are then used to create a ROM. This ROM is utilized for further analysis of the design.

5.1. Mathematical model and numerical solution

Heat exchanger fins are used to enhance removal of heat from any heated surface. Heat exchanger fins are of particular importance in cooling electronic components and other machinery and equipment. Changes in the shape of the fin can result in improved performance, thus reducing cost, space needed, and energy required for fans [4]. A number of researchers have studied the problem of heat dissipation from longitudinal fins, and earlier the authors developed an evolutionary algorithm coupled with numerical simulations to optimize the shape of heat exchanger fins [32]. The reader is referred to [32] for a more detailed discussion of this design problem. This section describes in brief the governing equations of momentum and energy as applied to the heat exchanger fin model, the numerical solution, and development of the ROM.

Fig. 5 shows a set of fins. Fluid (water) is pumped through the channel between the curved surfaces of two consecutive fins to remove heat. The direction of flow is along the positive z -axis. It is assumed that the flat plate is insulated and that the fluid velocity and temperature profiles are fully developed. The flow is also assumed to be laminar and incompressible, and any effects of natural convection are neglected. The whole system is assumed to be in steady state. Taking advantage of the symmetry, only one half of the fin is modeled, as shown in Fig. 6. The distance from the base of the fin to the insulated flat plate is assumed to be of unit length. The length of the fin is denoted by a , the base thickness by τ and the spacing between two consecutive fins by $2b$. The shaded portion represents the lateral surface of the fin. The thermal properties of the solid and fluid are assumed constant and only quadratic fin profiles are considered in this shape design problem. The mathematical model describing the fluid flow is given by

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = \frac{1}{\mu} \frac{\partial p}{\partial z} \quad (14)$$

The temperature distribution in the fin is

$$\frac{\partial^2 \theta_s}{\partial x^2} + \frac{\partial^2 \theta_s}{\partial y^2} = 0 \quad (15)$$

and the temperature distribution in the fluid is given by

$$\frac{\partial^2 \theta_f}{\partial x^2} + \frac{\partial^2 \theta_f}{\partial y^2} = \frac{\rho_f c_{pf}}{k_f} v \frac{\partial \theta_f}{\partial z} \quad (16)$$

The momentum equation has no-slip at the fluid-solid interfaces and symmetry boundary conditions on the other surfaces, respectively. The energy equations also have symmetry boundary conditions on the symmetric surfaces. At the solid-fluid interface, the boundary condition must satisfy the requirements of equality of temperatures and

temperature gradients. The above governing equations were solved numerically using the finite difference method. Further details on the discretization, grid generation, and numerical solution can be found in [32].

5.2. Shape design and ROM construction

The objective of the fin shape design problem discussed here is to study the velocity field and temperature distribution when the shape of the lateral surface of the fin is varied. With the temperature and velocity profiles known, the heat transfer coefficient, the pumping power, material cost, and manufacturing cost can be determined and considered as a part of the design space. In this study the base thickness of the fin and spacing between the fins can be varied in addition to the lateral surface being constrained to being quadratic in shape. A total of 15 designs chosen by sampling the allowable design space were used as the initial snapshot set. Table 1 shows some representative designs created by producers P_1 and P_2 . The shape design problem has five parameters, and the initial set of designs was chosen to account for all possible curvature variations of the lateral surface of the fin. The base thickness and fin spacing were chosen by randomly sampling the range of allowable values. Fig. 7 shows the distribution of the design space covered by the initial set. In Fig. 7, to represent the designs in three dimensions, the average of the three values of the points on the lateral surface of the fin is used in place of the actual values. It is possible that the sampling of the initial design set propagates error into the ROM, which can be overcome by adding and removing models as necessary from the ROM server. Also, if the initial snapshot set is insufficient to build an accurate ROM, more designs can be added to the ROM server as needed. A concatenated ensemble matrix with the temperature and velocities at each point in the discretized domain was constructed. The energy of the POD expansion is used to compute the accuracy of the ROM as described in Eq. (3). If the accuracy of the ROM is outside the acceptable range, more models are added to the snapshot set. To simulate the design process and the exchange of information between the producers and consumers, the following sequence of operations is adopted:

- 1) 15 designs were computed and used as the initial input to the ROM server
- 2) Consumers make use of the ROM server to evaluate designs and analyze data
- 3) More designs are added to the ROM server by the producers
- 4) All consumers seamlessly get updated information without making explicit updates and are able to evaluate designs that were previously not possible.

5.3. Information exchange and the ROM server

The ROM server eliminates the need for explicit information synchronization between producers and consumers during the design process. This section extends in detail the design process outlined in Section 4.2. The steps in the design process are simulated as shown in Fig. 8.

- 1) The producers P_1 and P_2 run the computational experiments needed to generate models for the POD ensemble matrix by varying the design parameters that are used to construct the ROM. Initially the ROM database holds the results from 15 computational models developed by the producers. Thus after this step the ROM server manages a database of 15 computational models, and the design space encompassed by this set of models is available to consumers (clients) to utilize for various tasks.
- 2) Evaluations of the ROM for design parameters not in the initial set of computational results are computed by the consumers, who also have the ability to request individual modes as well as the singular value spectrum from the ROM server. A consumer C_1 uses the ROM server to evaluate the velocity and temperature profiles for the following design parameters, which though not in the initial solution set lie within the design space of the problem. The input to the ROM server is the following list of parameters:

τ	b	y_0	y_1	y_2
0.160	0.260	0.150	0.100	0.195

The ROM server then performs the computation and returns the results to consumer C_1 as shown in Fig. 9.

- 3) Another consumer, C_2 , attempts to perform an evaluation for the following set of design parameters:

τ	b	y_0	y_1	y_2
0.210	0.220	0.190	0.105	0.060

This request for evaluation of a proposed design is not within the design space of the existing ROM based on the 15 computational models. This results in a non-evaluation and the return of an appropriate message to consumer C_2 and a notification to producer P_2 that additional analysis is needed to extend the design space. This may lead to a discussion between the consumers and producers about the direction of design, the design space, and the analysis goals.

- 4) Producer P_2 and consumer C_2 consult on the needed expansion of the design space, and producer P_2 performs the additional analysis and adds it to the ROM server. In some cases, for example, due to physical

constraints it might not be possible to extend the design space to accommodate the request from consumer C_2 .

- 5) The ROM server recomputes the ROM.
- 6) Producers P_1 and P_2 continue to populate and extend the design space with more models to the ROM server. The consumers can utilize the new information without explicit synchronization of the data. For example, the unsuccessful attempt to evaluate the model in step 3 can now be evaluated after additional models have been added to extend the design space. Thus the consumer C_2 can now request the ROM server to perform the same computation and visualize the normalized flow and temperature fields, as shown in Fig. 10.
- 7) Just as the POD-based ROM can be evaluated to determine the performance of a particular design configuration, consumers can also get information associated with the ROM, for instance the spectrum of singular values. The singular value spectrum shows the amount of energy associated with each dominant mode and is hence indicative of the number of modes to be considered in the POD approximation. Fig. 11 shows the variation of the singular value spectrum with the number of computational models utilized by the ROM server for the POD approximation. It is seen that as more computational models are added, the singular values begin to converge. In this case adding more than 15 computational models did not increase the accuracy of the ROM because the incremental energy captured by the later modes is only a small fraction of the total energy of the POD approximation. In this case the first five, ten, and fifteen modes capture 89.7%, 98.6%, and 99.99% of the total energy, respectively. Thus a ROM approximation based only on the first five modes will have a larger error compared to a ROM computed using the first fifteen modes. For a design engineer using a ROM to make engineering decisions, the singular value spectrum is an indication of the error (i.e., if the error is higher than the maximum acceptable value, more data needs to be added to the snapshot set for the ROM server to recompute an updated model). Because full CFD simulations are performed at the end of the design phase before finalizing a design, an appropriate cutoff of the number of modes can be made to enable design decisions at the early stages in the process.

By organizing and managing computational models in the ROM server, all members of an engineering team have access to the same information without repeated manual synchronization of data. It has also been demonstrated that each of the design tasks in the workflow can be accomplished using the POD-based ROM server, and the system

helps all consumers seamlessly get updated information without making explicit updates. This results in the organization saving time as well as serving the purpose of maintaining the version history of computational models.

6. Conclusions and future work

In this work a computational data management system is proposed that incorporates reduced-order modeling to enable a distributed framework for engineering design. The data management system has been enabled by developing a client-server based architecture. The server stores computational data in a vendor agnostic format and enables POD computations that can be used in analysis and optimization for simulation-based engineering design. The client-server based architecture enables persistent storage of computational data that can be accessed on-demand from any geographical location, thus improving collaboration among members of a distributed engineering team. The cost of synchronization was computed based on the size of the data and the number of times it needs to be transferred amongst multiple producers and consumers participating in a product design cycle, and this cost was computed for the cases with and without the ROM server. It was found that the cost of synchronization is lower when a ROM server is used in the design workflow. The software infrastructure developed allows real-time collaboration in a distributed engineering team by leveraging the fast computational capabilities of POD-based ROMs. As discussed above a batch process creates a bottleneck to information sharing, which is overcome by utilizing a server-based solution. Furthermore, the client-server architecture enables exposing a synchronized view of all existing computational models and also accounts for the changes to these models in a near real-time fashion. The centralized server-based solution also eliminates the user cost of data synchronization, which is high in a distributed engineering team. The solution developed in this work is especially useful to help with collaboration in a geographically distributed team of analysts and design engineers.

Further research is needed to address several issues. To assemble the ensemble matrix in the POD algorithm, all the CFD datasets are required to have the same number of grid points. This is a rather stringent requirement in most cases. Developing an algorithm to reconcile data from CFD models into the ensemble matrix can help automate construction of the ensemble matrix. In the client-server model because the computation is being performed on a remote server, the total computational time is the sum of the time to perform the arithmetic and the network communications time. The network communication time introduces latency, which was not considered in this article. This is an important aspect and needs further study. This current work, implicitly requires that the computational database be located on a single node because of the restrictions imposed by the ROM server. In the

future the authors plan to explore the possibility of utilizing a cloud-based ROM database to store as well as compute ROM parameters. The capabilities of the ROM server can thus be extended to incorporate multiple computational databases simultaneously. This will require that the ROM computation be performed in a distributed manner, where each node in the network can access data on other nodes based on some global metadata. Another aspect of this work that can be made more efficient is the recomputation of the POD basis functions when new data is added or removed from the ROM database. This can be improved by utilizing SVD update methods so that the ROM server is always available for processing requests. A lazy recomputation can be performed when the ROM server has a negligible client load. Another aspect of this work that needs further study is the development of appropriate clients that access and leverage the ROM server. For instance, an optimization client can be developed that utilizes the ROM server to evaluate the objective function. Client-side caching techniques can be developed to minimize the effects of latency in making network calls. Also, versioning of engineering models can help engineers keep track of and compare incremental changes during the design process.

Acknowledgment

This research was supported in part by the US Department of Energy – Office of Fossil Energy under Contract No. DE-AC02-07CH11358 through the Ames Laboratory.

References

- [1] Tanenbaum A, Steen M. *Distributed Systems: Principles and Paradigms*, Prentice Hall, 2002.
- [2] Kirby M. *Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns*, Wiley, 2001.
- [3] Suram S, McCorkle D, Bryden K. Proper orthogonal decomposition based reduced order model of a hydraulic mixing nozzle: AIAA MAO, Vancouver, Canada (2008).
- [4] Kays WM, London AL. *Compact Heat Exchangers*, 3rd ed, Krieger, 1998.
- [5] Samadiani E, Joshi Y. (2010) "Reduced order thermal modeling of data centers via proper orthogonal decomposition: a review", *International Journal of Numerical Methods for Heat & Fluid Flow*, Vol. 20 Iss: 5, pp.529 – 550.
- [6] Trefethen LN, Bau III D. *Numerical Linear Algebra*. SIAM, Philadelphia (1997).
- [7] Meyer CD. *Matrix Analysis and Applied Linear Algebra*. SIAM, Philadelphia (2000).
- [8] Tan BT. Proper Orthogonal Decomposition Extensions and Their Applications in Steady Aerodynamics. MS Thesis, Singapore-MIT Alliance (2003).
- [9] Bui-Thanh T, Damodaran M, Willcox K. Aerodynamics Data Reconstruction and Inverse Design using Proper Orthogonal Decomposition. *AIAA Journal*, vol. 42, No. 8.
- [10] Everson R, Sirovich L. Karhunen-Loève Procedure for Gappy Data. *Journal of the Optical Society of America*, 12(8):1657-1664.

- [11] My-Ha D, Lim KM, Khoo BC, Willcox K. Real-time optimization using proper orthogonal decomposition: Free surface shape prediction due to underwater bubble dynamics, *Computers and Fluids*, Vol 36, Issue 3: 499-512.
- [12] VTK, www.vtk.org, accessed on 1/27/2015.
- [13] Astrid P. Reduction of Process Simulation Models: a proper orthogonal decomposition approach. PhD Thesis, Technische Universiteit Eindhoven (2004).
- [14] Seymour K, Yarkhan A, Agrawal S, Dongarra J. NetSolve: Grid Enabling Scientific Computing Environments In *Grid Computing and New Frontiers of High Performance Processing*, vol. 14, *Advances in Parallel Computing* (2005).
- [15] Kozierok CM, *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference* (2005). No Starch Press. 1st ed.
- [16] Tannehill JC, Anderson DA, Pletcher RH. *Computational Fluid Mechanics and Heat Transfer*. (2nd ed.) Taylor & Francis (1997).
- [17] Mahule T et al.: PADMINI: A Peer-To-Peer Distributed Astronomy Data Mining System and a Case Study. *Conference on Intelligent Data Understanding* (2010).
- [18] Djilali S. P2P-RPC: Programming Scientific Applications on Peer-to-Peer Systems with Remote Procedure Call. *Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, pp.406. CCGrid (2003).
- [19] Hunter J, Choudhury S. Semi-automated preservation and archival of scientific data using semantic grid services. *Fifth IEEE International Symposium on Cluster Computing and the Grid*, vol. 1, pp.160 - 167. CCGrid (2005).
- [20] Sakurai T et al. A Hybrid Parallel Method for Large Sparse Eigenvalue Problems on a Grid Computing Environment Using Ninf-G/MPI. *Large-Scale Scientific Computing*. pp 438-445, Springer (2006).
- [21] Gunzburger MD. *Perspectives in Flow Control and Optimization*. Society for Industrial and Applied Mathematics (2002).
- [22] Kerschen G, Golinval JC, Vakakis AF, Bergman LA. The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems. *Nonlinear Dynamics*, vol. 41, pp 147-169 (2005).
- [23] Ly HV, Tran HT. Modeling and control of physical processes using proper orthogonal decomposition. *Mathematical and Computer Modeling*, vol. 33, pp. 223-235 (2001).
- [24] Chen J, Kostandov M, Pivkin IV, Riskin DK, Willis DJ, Swartz SM, Laidlaw DH. Visual analysis of dimensionality reduction for exploring bat flight kinematics in a virtual environment. *Joint Virtual Reality Conference of EGVE - ICAT - EuroVR* (2009).
- [25] Zhou X, Hitt DL. Proper Orthogonal Decomposition Analysis of Coherent Structures in Simulated Reacting Buoyant Jets. *AIAA Journal*, vol. 49, issue 5, pp. 945-952 (2011).
- [26] Korpela E. Distributed Processing of SETI Data: Searching for extra terrestrial intelligence. *The frontiers collection*, Part 2, pp 183-199 (2011).
- [27] Liu J. Micro-benchmark level performance comparison of high-speed cluster interconnects. *Proceedings. 11th Symposium on High Performance Interconnects* (2003).
- [28] Ghia U, Ghia KN, Shin C. T: High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method, *Journal of Computational Physics*, 48, pp 387-411, 1982.
- [29] Du, J., Zhu, J., Luo, Z. and Navon, I. M. (2011), An optimizing finite difference scheme based on proper orthogonal decomposition for CVD equations. *International Journal for Numerical Methods in Biomedical Engineering*, 27: 78–94.
- [30] Pahl, G., Beitz, W., Schulz, H.-J., Jarecki, U. Wallace, K.B., Lucienne T.M. (Eds.), *Engineering Design: A Systematic Approach*, 3rd Edition, 2007, Springer Verlag.

- [31] Ertas A, Jones JC. The Engineering Design Process 2nd ed, Wiley (1996).
- [32] Suram S, Ashlock DA, Bryden KM. (2006) Graph based evolutionary algorithms for heat exchanger fin shape optimization, Portsmouth, VA, United states: American Institute of Aeronautics and Astronautics Inc., 647-657.
- [33] Quinn M. Parallel Programming in C with MPI and OpenMP, 1st ed, McGraw-Hill (2005).

Fig. 1. Workflow in an engineering design.

Fig. 2. Workflow developed utilizing the ROM server.

Fig. 3. Information transfer and data synchronization; (a) single producer and consumer information transfer, (b) data synchronization between multiple producers and consumers, (c) synchronization with ROM server.

Fig. 4. Main components of the ROM server architecture.

Fig. 5. Schematic diagram of the fins.

Fig. 6. Single fin being modeled.

Fig. 7. Initial design space of heat-exchanger designs.

Fig. 8. Timeline of various producer/consumer interactions during the design process.

(a)

(b)

Fig. 9. Velocity (a) and temperature (b) profiles for the fin geometry $\tau = 0.160$, $b = 0.260$, $y_0 = 0.150$, $y_1 = 0.100$, $y_2 = 0.195$.

(a)

(b)

Fig. 10. Velocity (a) and temperature (b) profiles for the fin geometry $\tau = 0.210$, $b = 0.220$, $y_0 = 0.190$, $y_1 = 0.105$, $y_2 = 0.060$.

Fig. 11. Variation of singular value spectrum as a function of the number of computational models.

Table 1

Representative shape designs.

Producer	Base Thickness (τ)	Fin Spacing (b)	Points on Lateral Surface			Shape
			y_0	y_1	y_2	
P_1	0.118	0.224	0.126	0.115	0.210
P_1	0.011	0.334	0.326	0.115	0.210
P_1	0.020	0.134	0.086	0.085	0.075
P_1	0.200	0.219	0.186	0.103	0.052
P_1	0.05	0.315	0.280	0.203	0.251
P_2	0.150	0.250	0.050	0.100	0.210
P_2	0.181	0.270	0.250	0.100	0.189
P_2	0.030	0.450	0.360	0.320	0.360
P_2	0.121	0.420	0.213	0.312	0.213
P_2	0.142	0.320	0.183	0.183	0.183