

Software Tools to Enable Immersive Simulation

Felix Newberry · Corey Wetterer-Nelson · John A Evans · Alireza Doostan ·
Kenneth E Jansen

Received: mm/dd/yyyy / Accepted: mm/dd/yyyy

Abstract There are two main avenues to design space exploration. In the first approach, a simulation is run, analyzed, the problem modified, and the simulation run again. In the second approach, an ensemble simulation is performed and the battery of results is leveraged to construct a surrogate model for a given quantity of interest (QoI). The first approach allows a practitioner to methodically move through the design space and analyze a solution field. A disadvantage of this technique is that each new simulation requires time consuming setup. The second approach provides the practitioner with a global view of the problem, but requires *a priori* design space limits and the QoI specification. In this work we introduce an *immersive* simulation software framework that enables practitioners to maintain the flexibility of the first approach, while eliminating the burden of setting up new simulations. Immersive simulation can also be used to inform the second approach, establishing limits and clarifying QoI selection prior to the launch of an ensemble simulation. We demonstrate live, reconfigurable visualization of on-going simulations coupled with live, reconfigurable problem definition that guides users in determining problem parameters. Ultimately, an immersive simulation framework enables more efficient design space exploration that reduces the gap between simulations, data analysis and insight extraction.

Keywords HPC · Scientific visualization · Immersive simulation · *in situ* simulation · Computational steering

Felix Newberry · Corey Wetterer-Nelson · John A Evans · Alireza Doostan · Kenneth E Jansen
Ann and H.J. Smead Aerospace Engineering Sciences Department,
University of Colorado, Boulder, CO 80303, USA,
E-mail: kenneth.jansen@colorado.edu

1 Introduction

The solution of systems of partial differential equations (PDEs) provides scientists, engineers, and other practitioners with insight into a wide variety of physical phenomena. High performance computing (HPC) is the driving force behind increasingly accurate simulations that are capable of rapidly solving multi-physics, three-dimensional systems, or higher order approximations. The traditional simulation workflow for PDE solvers consists of sequential steps: meshing, definition of boundary conditions, solver and visualization. This structure leads to numerous inefficiencies. Problem redefinition and the role of visualization as a post-processing step delays engineering/scientific discovery and design. In HPC applications, the practitioner's pursuit of insight is hampered not only by time devoted to each new problem redefinition, but also by additional queue time. These traditional workflow inefficiencies inhibit design space exploration and subsequent ensemble simulation.

One method to address workflow inefficiencies is with notification and monitoring systems that query simulation data during runtime [39, 37, 44]. While this output does provide practitioners with basic solver behavior, such as convergence, it does not permit live reconfigurable simulation. Co-processing, the concept of performing *in situ* visualization and analysis while a simulation is ongoing, allows users to access simulation data immediately and mitigates the burden of writing large simulation data to disk [22, 46, 17, 29]. A summary of *in situ* methods, infrastructures and applications to HPC is presented in the state-of-the-art (STAR) report [7]. The authors observe that while numerous infrastructures are available, only a limited number of production quality frameworks have emerged. For instance, there are co-processing implementations within VisIt [11] using ADIOS [26], Lib-Sim [31], GLEAN [47], and in ParaView Catalyst (Catalyst) [1, 4]. A concluding remark in [7] is

that given the entry cost for new *in situ* frameworks is relatively high, a move towards a generic data interface would be of great value. More recently, attempts have been made to address this concern. The introduction of the *in situ* interface SENSEI [6] and infrastructure ALPINE [27] provides an interface between a given solver and a variety of visualization and analysis routines. While SENSEI is designed solely to support Catalyst, Lib-Sim and ADIOS infrastructures, ALPINE plays both the supporting role and provides its own execution model and visualization. In [5], a performance analysis of SENSEI shows the infrastructure is highly flexible and has low overhead.

A powerful extension of co-processing is computational steering, where users can take advantage of real time simulation feedback to rapidly explore the design space and extract insight. Computational steering has been previously demonstrated in the flow solver PHASTA with Catalyst [50, 54], though steering was accomplished via live edits of a solver parameter file, and geometric deformations were not implemented. More recently, Catalyst has been used to model turbidity currents [9], where steering is introduced through application specific LibMesh-sed-imentation code on solver parameters such as time step and tolerances. Another geological application utilized *in situ* visualization in earthquake modeling but did not use computational steering [33]. In [51], computational steering is enabled via a user interface, however this software is closed source. These examples show the current state of computational steering either requires additional software, the *in situ* editing of simulation solver parameter inputs or are closed source.

Ensemble simulation, informed via prior design space exploration, is a useful means of further insight extraction. Through visualizing and analyzing solution output across a range of different realizations of input parameters, practitioners can better understand how some output quantity of interest (QoI) is a function of input variables. The field of uncertainty quantification (UQ) addresses the challenge of mapping uncertain inputs to output QoIs [19, 28, 52]. Large ensembles allow the collection of useful statistics, creation of surrogate models for a given QoI, sensitivity analysis, and much more. Further, the evaluation of ensembles has traditionally been performed in serial, with parallel implementations that are predominantly *ad hoc*. While there has been some commercial software development in this area [2], in this work we employ the open source Python library libEnsemble, developed at Argonne National Laboratory as part of the Department of Energy Exascale Computing Project to coordinate the concurrent evaluation of dynamic ensembles of calculations on massively parallel resources [25]. In [10], libEnsemble is coupled with multiobjective optimization to achieve better resource utilization for HPCs. Having performed an ensemble simulation, the practitioner may access one of several open source UQ

Python libraries such as Uncertainpy [45], chaospy [16], and UQ-PyL [48]. Additional Python libraries for specific UQ fields such as active subspaces are also in use [12]. The priority research directions for *in situ* data management identified by the authors in [36] include ensemble analysis, UQ, and surrogate models. We note that there is a distinction between random input variables addressed by UQ and design variables examined in design space exploration. In this work we apply UQ methodologies to design variables by treating them as uniform random variables.

1.1 Contributions of this work

While numerous studies have demonstrated co-processing, to the authors' knowledge, the valuable link between co-processing capabilities, extraction of relevant data from ensemble generation, and subsequent application of UQ techniques such as sensitivity analysis, has not been explored. In this document, we introduce an immersive simulation software framework that enables more rapid design space exploration with computational steering and subsequently informs ensemble generation for more efficient insight extraction.

Our work builds directly upon previous implementations of Catalyst in the flow solver PHASTA [54, 4]. Where earlier demonstrations of co-processing functioned with Catalyst hooked directly into the PHASTA solver, our work differs in that we implement SENSEI to interface between the solver and Catalyst. The main contribution of this work is to introduce and demonstrate an immersive simulation software framework. We describe the relationship between a PDE solver and immersive simulation software components that enable computational steering and modification of physical, geometric and solver parameters. We provide software linkages that others can leverage or modify as needed. Equipped with an immersive simulation framework, a practitioner may interact with their simulation *in situ* via the ParaView graphical user interface (GUI).

We complement these immersive simulation tools with a demonstration of libEnsemble for straightforward ensemble simulation and chaospy to implement UQ methods to analyze the ensemble results. We show the ability of immersive simulation and ensemble tools to complement and enhance one another.

The remainder of this paper is organized as follows. Section 2 describes the simulation workflow, first addressing the traditional approach in Section 2.1 followed by an immersive simulation workflow in Section 2.2. Next, Section 3 details the software elements that make up an immersive simulation software framework. Section 4 provides a numerical demonstration of immersive simulation tools on a 2D aggressive subsonic diffuser with the fluid solver PHASTA. Section 5 describes ensemble simulation software tools before

Section 6 applies the ensemble tools, coupled with an immersive simulation framework, to a global sensitivity analysis. Finally, Section 7 summarizes this study's conclusions.

2 The simulation workflow

We now describe the traditional simulation workflow, highlighting aspects that inhibit scientists' and engineers' ability to gain insight from their simulations. We then present an immersive simulation workflow that alleviates standard workflow inefficiencies.

2.1 Traditional simulation workflow

The traditional simulation workflow, whether a lone simulation (Figure 1 (a)), or an ensemble simulation (Figure 1 (b)), consists of a series of sequential steps. Considering Figure 1 (a), we refer to components of the workflow that occur before and after the solver step as belonging to the pre- and post-process, respectively. First, in the pre-process we develop an idea into a solvable problem through defining an appropriate geometry, mesh, and boundary conditions. Next, we calculate the solution to our problem through our chosen PDE solver. Finally, in the post-process section we perform visualization and data analysis to determine the utility of the solution data. If the present results are satisfactory, then we have obtained the insight we seek and the simulation workflow is complete. Alternatively, if the results are unsatisfactory and we want to redefine the problem to explore the design space, then we return to the pre-process region of the workflow.

Difficulties with the traditional simulation workflow arise due to the computational expense of individual workflow components coupled with the persistent need for problem redefinition. Solution data may persuade a practitioner to return to the pre-process steps either to ask a new question of the simulation, or to improve the method of answer to an existing question. As an example, let us consider the visualization of a fluid flow. Inspection of high gradient regions in the flow, such as boundary and shear layers, may reveal that further mesh refinement is needed to accurately resolve flow behavior. Alternatively, the practitioner may decide geometric adjustments would be a productive line of inquiry and be motivated to update the geometry and associated mesh. The role of problem redefinition as a post-process step in the traditional workflow delays engineering and scientific insight.

A vital concern for HPC applications that will grow with the arrival of exascale computing is the bottleneck of IO. The PDE introduced in [15], in which an unsteady flow problem is addressed, was scaled up to 3.1 million processes and 92 billion elements in [38]. To save the complete solution of a simulation this size would generate $O(1)$ ter-

abyte of data per second. Although the code can determine a write frequency and produce numerous flow statistics, analysis of unsteady flow structures is not practical with the traditional simulation workflow. To undertake post-processing steps that involve writing, and re-reading of data at this rate is unreasonable. Additionally, we find conventional methods impractical even if we consider specialized libraries that achieve close to machine IO bandwidth limit (240 Gb/s, or more than 4 seconds to write 1 second of simulation data) [18, 30].

Both the aforementioned traditional workflow concerns, i.e., the delay of insight caused by problem redefinition and impracticality of visualization in HPC applications, are exacerbated when we consider ensemble simulation; see Figure 1 (b). Catering to an array of pre-process settings and tailoring post-process steps to each ensemble member is cumbersome for the practitioner. Additionally, the traditional workflow delays the initial setup of an ensemble due to the problem redefinition necessary to figure out design space limits and adjust a given QoI.

2.2 Immersive simulation workflow

An immersive simulation workflow intends to mitigate some of the standard workflow inefficiencies through live, reconfigurable simulation. In essence, a new co-process stage integrates steps from both the pre-process, such as problem definition, and post-process, such as visualization and data-analysis, to be contemporary with the ongoing/live advancement of the flow solver; see Figure 2.

Co-processing methods aim to vastly reduce the quantity of data storage through *in situ* visualization that the user is able to interact with and calibrate to their specific needs [22, 46]. In co-processing, the visualization pipeline is shifted from being a post-processing step within the PDE solver workflow to be concurrent with the simulation. The progression of HPC from present petascale to future exascale computing, coupled with corresponding increase in simulation complexity, such as multi-scale and multi-physics, further enhance the desire for co-processing as opposed to traditional post-processing capabilities.

Immersive simulation couples co-processing that allows live visualization with computational steering that allows live problem redefinition. When immersed, a practitioner is capable not only of visualizing their problem solution *in situ*, but also of intuitively redefining the problem definition for rapid exploration of the design space. The practitioner may choose to steer solver parameters, boundary conditions, and even the problem geometry depending on what aspects of the problem space engage them. In the HPC setting, an immersive simulation practitioner can evade the queue time associated with each new problem redefinition. In this manner, the behavior of different combinations of input param-

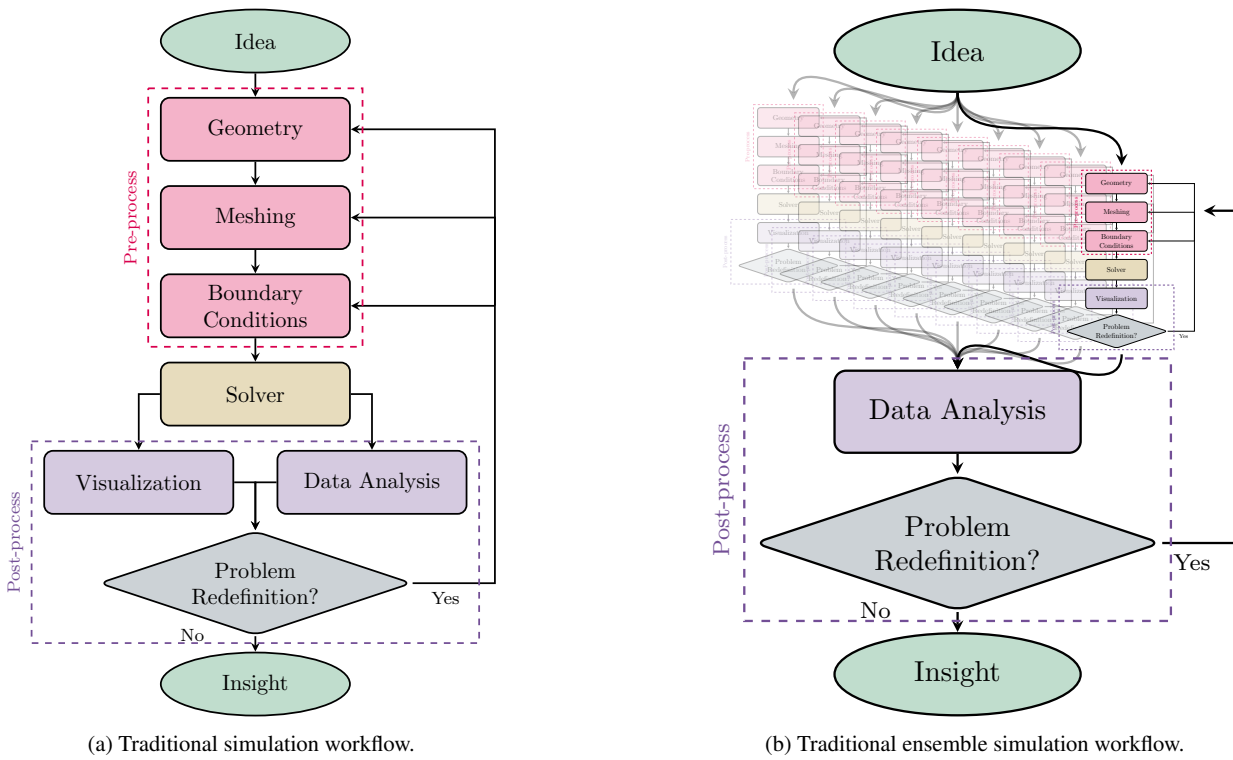


Fig. 1: Traditional singular (a) and ensemble (b) simulation workflows.

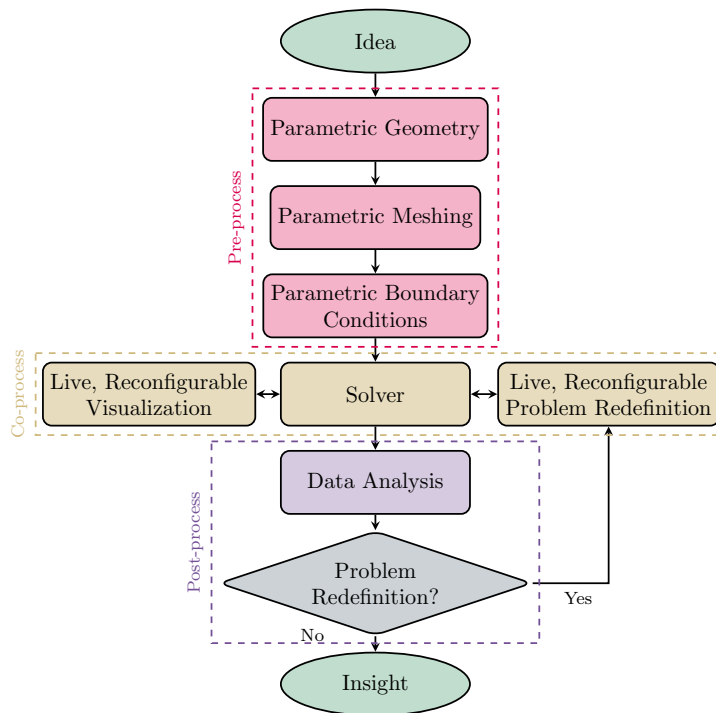


Fig. 2: Immersive simulation workflow.

ters can be quickly understood and inform appropriate limits for subsequent or concurrent ensemble simulation.

3 Software tools for immersive simulation

In this section we describe the core components of an immersive simulation software framework; see Figure 3. This framework enables multiple forms of immersive simulation that work with a traditional PDE solver to perform both live, reconfigurable visualization and live, reconfigurable problem definition. The colored boxes indicate software components while the grey boxes show functionality. Thin arrows out of the *in situ* infrastructure *Catalyst*, or interface *SENSEI*, represent data extraction/compression, while thin arrows into these components relay analyst input via the *ParaView* GUI. Large data streams, shown by thick arrows, correspond to the full solution state (problem mesh and solution). The analyst obtains insight, shown by a wide arrowhead, through engagement in an immersive simulation through the *ParaView* GUI. A crucial aspect of immersive simulation is that the commands related by the analyst to reconfigure the simulation are conveyed with small data streams. With this setup, an analyst can operate the live visualization and problem definition for swift simulation feedback and insight extraction. We now provide some context on the integration of core immersive simulation software components *SENSEI*, *Catalyst*, and *ParaView* with the PDE solver. While we use the fluid solver *PHASTA*, many aspects of the links between software and all of the resulting advantages are general. We note that *SENSEI*, *Catalyst*, and *ParaView* are all open source products of Kitware.

3.1 *SENSEI in situ* interface

SENSEI is a generic *in situ* interface best summarized by the underlying premise of “write once, use everywhere” [6]. The interface supports a number of *in situ* infrastructures such as *VisIt/Libsim*, *Catalyst*, and *ADIOS*. Simulation output is mapped to the VTK data model via a data adaptor and an analysis adaptor, while an *in situ* bridge links the respective adaptors and prompts *in situ* analysis. An in depth description of *SENSEI* is provided in [6] as well as a brief view from the simulation code, and the perspective of different choices of *in situ* infrastructure. In the *SENSEI* repository mini-apps are provided as a guide for different PDE solver applications. The mini-app topics span *SENSEI* Python bindings, C++ solvers for dynamic periodic oscillators, the Mandelbrot set and vortex simulation, and the *PHASTA* mini-app that we build upon for this work.

In general the linking software to enable immersive simulation appears in three places. First, the *SENSEI* mini-app

bridge must be updated to match any parameters the practitioner seeks to pass from their PDE solver. As shown in [6], the bridge is custom to the implementation code and requires initialization, analysis, and finalization steps. In this work, we focus on how this bridge communicates with the PDE solver from the solver’s view, and also on the practical implementation in terms of necessary scripts at runtime. From the perspective of the *PHASTA* solver the abbreviated simulation code to highlight the *SENSEI* footprint is presented in Figure 4. Within the simulation script we make calls to subroutines that initialize, co-process and finalize the interaction of the interface with the solver.

These subroutines are defined in a second PDE solver script *sensei_interface.f* shown in Figure 5. Figure 5 essentially refines the main simulation routine calls to *SENSEI* by providing additional context. The routines called here are defined in the *SENSEI PHASTA* bridge. For instance, the initialization subroutine calls any *in situ* analysis adaptor, such as *Catalyst*, before communicating mesh details vital to live visualization and a set of fields determined by the user.

For the full context of these scripts we refer the interested reader to our GitHub repository www.github.com/SimNautilus/nautilusFlow that documents the links between the PDE solver *PHASTA* and *in situ* interface *SENSEI*. In our implementation of an immersive simulation framework we initially setup the main solution variables, such as mesh coordinates, velocity and pressure fields, for live visualization. Next, we improve the framework with the introduction of steering parameters or more complex fields that are passed to *SENSEI*.

Having setup the PDE solver to communicate fields to *SENSEI*, we next address the implementation at runtime. As noted, several *in situ* infrastructures are supported. The practitioner can setup the desired analysis type with ease in the equivalent of *senseiPhasta.xml*; see Figure 6. This script is called via *sensei_adaptors.init()* during the *SENSEI* initialization steps in Figure 5. While the *SENSEI* bridge is created during initialization, the analysis it facilitates, for instance via *Catalyst*, is reconfigurable during a simulation run.

3.2 *Catalyst in situ* infrastructure

Catalyst, formally the *ParaView* co-processing library [15], is an *in situ* visualization library with an adaptable API that is built on VTK [40] and *ParaView* [1]. Through building the library with VTK, *Catalyst* can access a plethora of useful algorithms such as IO writers, visualization filters, and graphics rendering. The visualization pipelines can be described in C++ or Python, and the *ParaView* GUI facilitates the generation of *Catalyst* co-processing Python scripts both directly, and via a Python trace and shell. Additionally, *Catalyst* is designed to connect visualization pipelines remotely through server-client architecture. While the addi-

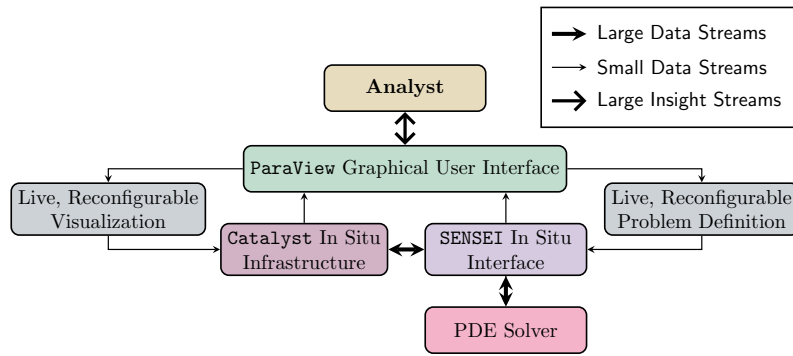


Fig. 3: Immersive simulation software framework.

```

subroutine itrdrv( ... )
  call ph_sensei_init(...)
  ! advance simulation
  call ph_sensei_coprocess(...)
  ! advance simulation
  call ph_sensei_fini()
  ! cleanup
end

```

Fig. 4: Pseudocode of the SENSEI footprint in PDE solver simulation script.

```

subroutine ph_sensei_coprocess(...)
  ! read in fields from solver
  call sensei_coprocess(...)
end
subroutine ph_sensei_init(...)
  call sensei_adaptors_init()
  call createpointsandallocatecells(...)
  ! iterate PDE solver partitions
  do i_block=1, n_blocks
    ! count vertices per element,
    ! elements in block,
    ! and pass to SENSEI
    call insertblockofcells(...)
  enddo
  call addfields(...)
end
subroutine ph_sensei_fini()
  call sensei_adaptors_fini()
end

```

Fig. 5: Pseudocode of the PDE solver link between simulation script in Figure 4 and SENSEI.

tion of the Catalyst adapter to solvers must to some degree inhibit solver performance, this influence has been shown to be negligible when compared to the savings in file I/O and data management [15]. In our numerical example we implement Catalyst with the fluid solver PHASTA via the

```

<sensei>
  <!-- A simple Catalyst Analysis -->
  <analysis type="catalyst"
    pipeline="slice"
    array="pressure"
    association="point"
    image-filename="pressure-%ts.png"
    image-width="1920"
    image-height="1080"
    slice-normal="0,0,1"
    enabled="0" />
  <!-- Catalyst co-process script -->
  <analysis type="catalyst"
    pipeline="pythonscript"
    filename="/path_to_file/
      my_catalyst_pipeline.py"
    enabled="1" />
</sensei>

```

Fig. 6: Pseudocode of the runtime XML description that configures *in situ* analysis routines [6].

SENSEI *in situ* interface, specifically through analysis adaptor in Figure 6.

A powerful function of Catalyst that has been alluded to is the setup of ParaView pipelines. The entire pipeline script can be generated intuitively from a ParaView GUI session that writes out a Catalyst co-processing script as shown in Figure 7. The co-processing script can be tailored to specific problems and is loaded automatically when live visualization is launched to facilitate the user's analysis. The practitioner can update the pipeline live through the Catalyst connection in the ParaView GUI.

Having described the implementation of the interface SENSEI and infrastructure Catalyst we are equipped with an immersive simulation framework of Figure 3. With these immersive simulation tools in hand a practitioner can leverage interactive visualization and problem redefinition for efficient design space exploration.

```

# ----- CoProcessor definition -----
def CreateCoProcessor():
    def _CreatePipeline(...):
        class Pipeline:
            # ParaView pipeline
            # Sequence of filters
            return Pipeline()
    # ----- Processing method -----
    def DoCoProcessing(...):
        # Update coprocessor with newly
        # generated simulation data
        # If the pipeline hasn't been setup
        # yet, this will set it up.
        coprocessor.UpdateProducers(...)
        # Write output data,
        # if appropriate.
        coprocessor.WriteData(...);
        # Write image capture,
        # if appropriate.
        coprocessor.WriteImages(...)
        # Live Visualization, if enabled.
        coprocessor.
            DoLiveVisualization(...)

```

Fig. 7: Pseudocode of the Catalyst co-processing script that enables *in situ* visualization and loads a chosen ParaView pipeline [4].

4 Numerical demonstration: immersive simulation tools

We demonstrate the utility of immersive simulation tools on a 2D aggressive subsonic diffuser simulated in PHASTA [50]. In this section we first describe the diffuser model equipped with parametric geometry deformation and then illustrate the application of immersive simulation tools described in Section 3. Code examples of the implementation of immersive simulation tools in PHASTA can be found in our GitHub repository

www.github.com/SimNautilus/nautilusFlow.

4.1 2D aggressive subsonic diffuser

Aircraft frequently require air intake systems that reroute air from the free stream velocity and slow it to speeds appropriate for the engine [3]. The intake arrives at the compressor on the aerodynamic interface plane (AIP). It is desirable that the AIP air exhibits low swirl, low distortion, and high pressure recovery, with minimal increase of the intake drag, lest engine performance is inhibited [32]. Therefore, understanding of intake flow is imperative to effective simulation of engine performance.

Typical intake geometries consist of a duct to direct the flow connected to a diffuser that, via an increase in cross-sectional area, trades upstream kinetic energy for a rise in downstream static pressure. This increase in area produces

an adverse pressure gradient that encourages separated flow. Furthermore, the flow dynamics may be complicated by duct curvature that introduces spanwise pressure gradients.

The diffuser model we study is motivated by a 3D transonic diffuser with two significant simplifications made in order to reduce computational expense: the model is a 2D slice of the 3D geometry midplane and the flow is treated as incompressible as opposed to compressible. Shifting from 3D to 2D greatly reduces the number of elements in the simulation mesh, and treating the flow as incompressible permits a much larger time step while maintaining a converging simulation. We emphasize the severity of the model simplifications on the diffuser; the purpose of this simulation is to demonstrate an immersive simulation software framework and to capture quantitatively the influence of upper and lower blowers on core flow behavior.

The diffuser geometry, depicted in Figure 8, is considered compact with a length to diameter ratio of $L/D \sim 1.03$ and a high expansion ratio of $ER = 2.2$. Active flow control is implemented through two tangential blowers, denoted as the upper blower (UB) and lower blower (LB). The blowers are designed to inhibit or delay flow separation on their respective surfaces. We prescribe a trapezoid waveform inflow condition to both the UB and LB, where we set the wave's mean, amplitude, and period. The time the trapezoid wave spends rising, falling, and at the maximum value of the wave is set to $1.5e-3$ s, $1.5e-3$ % and $4e-3$ s, respectively. We have fixed the period to $8e-3$ s for the study and ensured a temporal resolution of 80 time steps for each blower period with a time step of $dt = 1e-4$ s.

The incompressible flow through the diffuser is modeled with an inflow Mach number $M_\infty \approx 0.052$ and bulk Reynolds number $Re_b \approx 4e4$. This Reynolds number is similar to the 2D plane diffuser experiments [34, 8]. The computational mesh made up of 241k tetrahedral elements is shown in Figure 8. The mesh is only one element deep (into the page as displayed in Figure 8), and periodic boundary conditions are employed in the depth direction to model 2D flow. Every boundary layer has first point off the wall is set to $2e-6$ m and a stretching ratio of 1.25 to maintain $y^+ < 1$ at the wall boundaries. The mesh is further refined at the entrance of the blowers and downstream to resolve the recirculating region.

In this study, we examine the QoI measuring the recirculating region length defined as

$$\ell_r = \sqrt{\int_{\bar{u}_x < 0} dx d_{\text{wall}}}, \quad (1)$$

where \bar{u}_x denotes the time-averaged stream-wise velocity and d_{wall} the distance from the wall. We integrate with respect to d_{wall} as opposed to dy to penalize recirculating regions that extend further into the core body of the flow. In the case where separation occurs in both the lower and up-

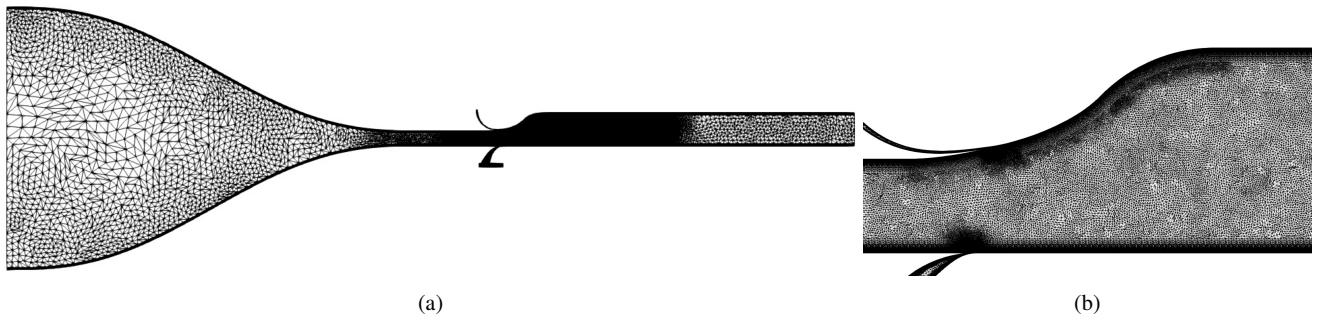


Fig. 8: 2D aggressive subsonic diffuser geometry (a) entire geometry, (b) close up of the blower control region showing the UB and LB.

per walls, we add the respective integrals together prior to taking the square root to find the length.

4.1.1 Parametric geometry deformation

In order to explore the effect of UB location along the curved upper extent of the diffuser, a custom parametric mesh modification tool was developed which slides the UB tangentially along the curved surface, moving the mesh vertices smoothly, allowing for geometric design exploration *in situ* without the need for re-meshing or multiple hand-generated CAD configurations. This procedure relies on Sederberg’s freeform deformation algorithm [41] with a geometry-fitted collection of 493 background non-uniform rational B-Spline (NURBS) patches deformed in unison with a custom geometry management algorithm. The geometry modification is driven by a single parameter φ varying continuously from 0 to 1 representing locations far down the curve of the diffuser and past midway up the diffuser as depicted in Figure 9 (b). These extents were chosen to limit mesh distortion under geometry modification. Figure 9 (a) illustrates the collection of background NURBS patches already deformed to the $\varphi = 1$ extent.

This geometry deformation algorithm was custom designed for the diffuser problem, employing low level OpenCASCADE commands. The general procedure here is extensible to other 2D geometric considerations. Further, this procedure is a proof of concept for how geometric variables can be included in immersive simulation design space exploration procedures. Future improvement to this procedure may incorporate higher level abstractions for parametric geometry modification such as the programmable CAD system Engineering Sketchpad [21]. More general tools for geometric modification to HPC applications are presented in [49].

4.2 Computational steering

We employ Catalyst, SENSEI, and the ParaView GUI to implement live visualization and computational steering on the 2D diffuser. The parameters for which we enable steering are the mean and amplitude of the UB and LB velocities, the UB position parameterized between $\varphi = 0$ and $\varphi = 1$, as shown in Figure 9 (b), and the weight term w , useful for assessing time averaged quantities. The QoI we are interested in, the recirculating region defined in Section 4.1, is measured as a time average to account for the transient boundary conditions on the UB and LB. We update the time averaged stream-wise velocity with

$$\bar{u}_x = wu_x + (1 - w)\bar{u}_x^0,$$

where u_x is the present time step’s stream-wise velocity and \bar{u}_x^0 denotes the previous time step’s averaged stream-wise velocity. Increasing w weighs recent steps more heavily and “shortens” the time-average interval, while decreasing w reduces the weight of new time steps and “lengthens” the time-average interval. In general, we set $w = 0.01$. In this manner, we can calibrate the memory of the time-averaged velocity field.

To equip PHASTA with the steering parameters the chosen fields must be passed from the PHASTA solver to SENSEI, and the SENSEI PHASTA adaptor updated for these fields as described in Section 3. Once implemented, the steering parameters can be interacted with during a Catalyst live visualization; see Figure 10. The user can adjust a given steering parameter, for instance the UB position, and immediately see how this alteration impacts the flow behavior. In Figure 10 we see a complex Catalyst VTK pipeline that determines regions in the diffuser flow with negative time averaged stream-wise velocity and calculates the associated recirculation length. Note, the practitioner’s interaction with the ParaView GUI allows them to create, visualize, and explore different VTK pipelines and easily examine other QoIs. In this manner, the practitioner can directly observe the effect of variations in key parameters on their QoIs, gain in-

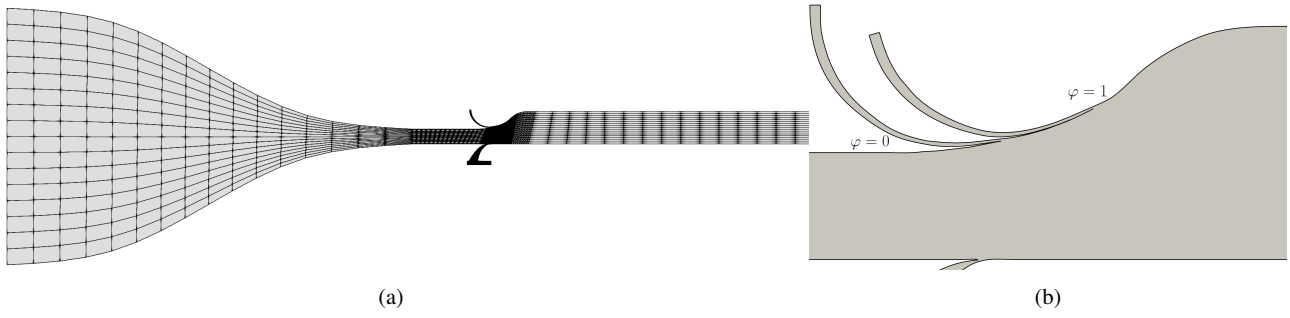


Fig. 9: Parametric 2D aggressive subsonic diffuser geometry: (a) spline track used to guide the deformation of the background NURBS surfaces (b) close up of the blower control region showing blower position set to $\varphi = 0$ and $\varphi = 1$.

sight more rapidly, and do so without saving the visualized flow data to disk.

We leverage computational steering to clarify QoI selection and setup design space limits for subsequent ensemble simulation. Our recirculation length QoI, defined in Section 4.1, initially accounted for the upper recirculation area alone and integrated over dy rather than d_{wall} . Computational steering allows us to observe the lower separation area and trial integration with respect to distance from the wall. Live adjustments of the blower control steering parameters suggest that low values of UB velocity mean, i.e., $< 1.43\%$ total diffuser mass flow rate, produce negligible reduction in the upper recirculation area. High values, $> 2.85\%$, result in fully attached flow, indicate additional velocity would be a poor use of flow mass. Similar testing of the LB mean produces an interval of $[0.00, 2.34]$. The LB mean is useful at small percentages too because the diffuser geometry means that lower separation occurs only when the core flow is pulled up by large UB action. This tug of war effect can be seen in Figure 10. Finally, we decide to examine the interval $[0, 1]$ for blower position φ . While a higher blower position improves this QoI, our study here is 2D and incompressible, hence we stay conservative in our design space constraints.

5 Software tools for ensemble sensitivity analysis

In Section 4.2, we employed immersive simulation to explore the design space of the 2D diffuser problem and to determine a QoI. Our next objective is to leverage this information to spawn an ensemble simulation. We first, in Section 5.1, provide background on global sensitivity analysis. Next, in Sections 5.2 and 5.3 we present software useful for efficient ensemble generation and the application of UQ techniques, respectively.

5.1 Sensitivity analysis

Parameter sensitivity is central to achieving an optimal design and performing UQ. Through visualization of a solu-

tion subject to deviations from nominal parameter values in boundary conditions, material properties, or geometric parameters, a user can navigate the design space with greater efficacy. In variance-based sensitivity analysis, the variance of a given QoI is decomposed into fractions that are attributed to specific input parameters, or interactions of those parameters. Monte Carlo simulation is a popular candidate for obtaining sensitivity metrics but can become infeasible for computationally demanding models. A suitable alternative is to use a UQ technique to approximate the QoI with an expansion in multivariate orthogonal polynomials, known as the polynomial chaos (PC) expansion [19, 53]. We next describe PC expansions followed by their application to global sensitivity analysis. We note that our PC expansion is described with random input variables general to the UQ field. Our implementation of the PC expansion treats design variables as uniform random variables.

5.1.1 Polynomial chaos expansions

We consider the scalar QoI, $u(\mathbf{\Xi})$, assumed to have finite variance, as a function of a d -dimensional vector of random inputs $\mathbf{\Xi} := (\Xi_1, \dots, \Xi_d)$ with joint probability density function $f(\mathbf{\xi})$. The PC expansion approximates the QoI as

$$u(\mathbf{\Xi}) = \sum_{j=1}^{\infty} c_j \psi_j(\mathbf{\Xi}),$$

where $\psi_j(\mathbf{\Xi})$ is a multivariate orthogonal polynomial evaluated at the random inputs and weighted by deterministic coefficients c_j . The polynomials $\psi_j(\mathbf{\Xi})$ are chosen to be orthogonal with respect to the probability measure $f(\mathbf{\xi})$. For instance, if $\mathbf{\Xi}$ follows a jointly uniform or Gaussian distribution, then $\psi_j(\mathbf{\Xi})$ are multivariate Legendre or Hermite polynomials, respectively [53]. We assume $\psi_j(\mathbf{\Xi})$ are normalized such that $\mathbb{E}[\psi_j^2(\mathbf{\Xi})] = 1$, where $\mathbb{E}[\cdot]$ represents the mathematical expectation operator. The expansion is truncated to a finite number of terms as

$$u(\mathbf{\Xi}) \approx \sum_{j=1}^P c_j \psi_j(\mathbf{\Xi}). \quad (2)$$

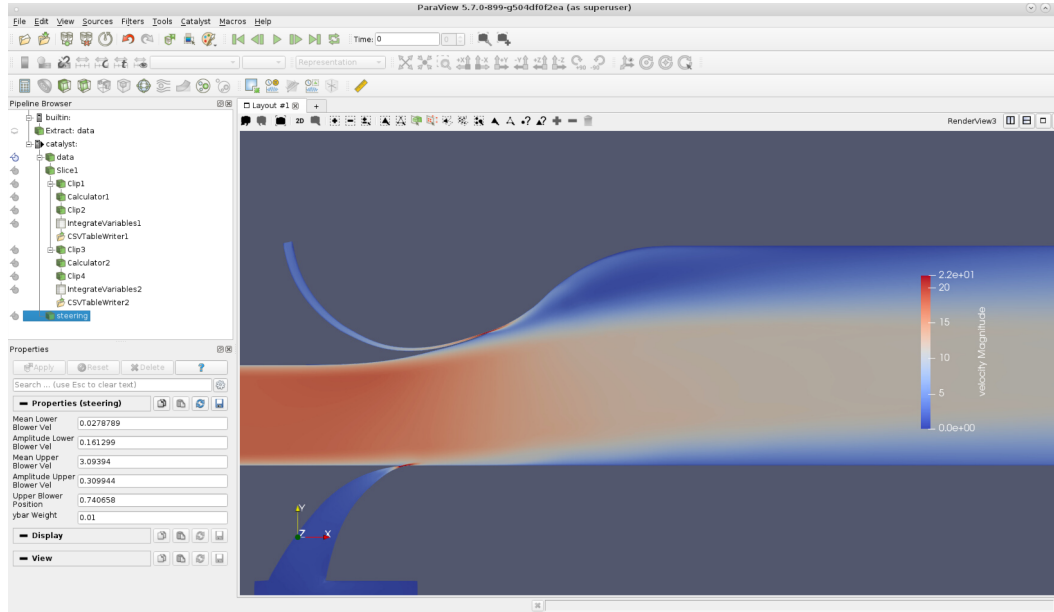


Fig. 10: ParaView GUI displaying the Catalyst live visualization and SENSEI computational steering. On the left, the pipeline browser shows the visualization pipeline and steering parameters.

An expansion with total order p and dimension d has $P = \frac{(p+d)!}{p!d!}$ terms. As $P \rightarrow \infty$, for a sufficiently smooth $u(\Xi)$, the PC expansion converges in the mean-square sense to u . The PC coefficients can be used to determine QoI statistics, construct a surrogate model, or perform sensitivity analysis. For instance, the QoI mean and variance can be computed from the PC expansion coefficients as $\mu_{PC} = c_1$ and $\sigma_{PC}^2 = \sum_{j=2}^P c_j^2$, respectively. To identify PC coefficients $\mathbf{c} = (c_1, \dots, c_P)^T$ in (2) we generate realizations of Ξ denoted by ξ . We seek to solve for \mathbf{c} in the linear system from N independent samples of ξ as

$$\mathbf{u} \approx \Psi \mathbf{c}, \quad (3)$$

where

$$\Psi(i, j) := \psi_j(\xi_i) \text{ and } \mathbf{u} := (u(\xi_1), \dots, u(\xi_N))^T. \quad (4)$$

If the resulting regression problem is over-determined, with $N > P$, we employ least squares approximation [23, 20], whereas if it is under-determined, with $N < P$, we apply compressed sensing [14, 35, 24, 13]. Compressed sensing methods are advantageous because they reduce the number of sample evaluations to determine PC coefficients.

5.1.2 Global sensitivity analysis via PC expansions

The first order Sobol' indices, S_{Ξ_k} , measure the effect of varying one parameter alone, Ξ_k , averaged over the variations in other parameters. Total indices, $S_{\Xi_k}^T$, refer to the variation in the QoI accounted for by one parameter and any interactions it has with other input parameters. In this work,

we apply global sensitivity analysis analytically as a post-processing of PC expansion coefficients [42]. The first order Sobol' indices are given by

$$S_{\Xi_k} = \sum_{i \in I_k} \frac{c_i^2}{\sigma_{PC}^2}, \quad (5)$$

where, I_k is a set of indices for which $\psi_j(\Xi)$, $j = 1, \dots, P$, is a function of only Ξ_k . Similarly, the total Sobol' indices are given by

$$S_{\Xi_k}^T = \sum_{i \in I_k^T} \frac{c_i^2}{\sigma_{PC}^2}, \quad (6)$$

where, I_k^T is a set of indices for which $\psi_j(\Xi)$, $j = 1, \dots, P$, is a function of Ξ_k and any other inputs.

5.2 Ensemble generation via libEnsemble

In this work, we employ `libEnsemble`, an open source Python library developed at Argonne National Laboratory as part of the Department of Energy Exascale Computing Project to coordinate the concurrent evaluation of dynamic ensembles of calculations on massively parallel resources [25]. `libEnsemble` aims to achieve extreme scaling, resilience, task monitoring and resource recovery, portability and flexibility, and exploit persistent data flow. The ensemble is coordinated via a manager/worker scheme that operates on several communication protocols.

`libEnsemble` is implemented through four components. The generator function `genf` produces values for simulations, the simulator function `simf` performs a simulation for given values from `genf`, the allocation function determines whether `genf` or `simf` should be called, and the calling script defines the parameters for the previous three functions and executes `libEnsemble`. The `libEnsemble` user manual [25] provides more detail on each of these components, and various tutorials. In Figure 11 we show an abbreviated version of the calling script. Note, in our application we have no generator function because `simf` reads inputs directly from a file. In the calling script we specify the number of workers, and the number of simulations. For example, an HPC job may use 50 nodes in total, but require 2 nodes per simulation. In this case, the number of workers would be 25. In Figure 11 we first setup the PHASTA executable. Next we create an ID number for each simulation before defining parameters for the simulation and generation functions. An exit criteria is specified to terminate the ensemble when the maximum number of simulations is reached. Finally, the ensemble is launched. As simulations complete and workers become available new simulations are launched. Other ensemble manager tools are also available, for instance, UQ Pipeline [43].

5.3 Sensitivity analysis with chaospy

We use the Python library `chaospy` to perform UQ PC based sensitivity analysis [16]. More generally, `chaospy` includes a suite of UQ methods such as low-discrepancy sampling, quadrature creation, polynomial manipulations, among others. A collection of Jupyter notebooks lets users explore the variety of `chaospy` applications.

We present pseudocode of the `chaospy` enabled sensitivity analysis in Figure 12. The main idea is to construct a PC expansion and use the coefficients to calculate Sobol indices following Sections 5.1.1 and 5.1.2, respectively. We note that in this pseudocode we do not include the validation error. In practice, we recommend reserving $\approx 20\%$ of samples to compute the PC expansion validation error, and use this to calibrate the polynomial order p .

6 Numerical demonstration: ensemble simulation tools

We have now, courtesy of an immersive simulation framework, established design space limits and confidence in our choice of QoI. We are also equipped with software tools to perform ensemble simulation and analyze the results. In this section, we demonstrate the utility of `Catalyst` during the collection of ensemble data, and carry out a global sensitivity analysis on the 2D aggressive subsonic diffuser.

```

import numpy as np
# Import libEnsemble content
from libensemble.libE import libE
from libensemble.executors.
    mpi_executor import MPIExecutor
from libensemble.tools
    import parse_args

# Import phasta sim_f and alloc_f
from phasta_sim
    import phasta_evaluate as sim_f
from allocation_func
    import phasta_alloc as alloc_f

nworkers = 25
n_sim = 50

sim_app = path_to_file / phasta.exe
exctr = MPIExecutor(central_mode=
    True)
exctr.register_calc(full_path=sim_app,
    calc_type = 'sim')

# Set up H0 - used as run_ID
H0 = np.zeros(n_sim, dtype=[('x',
    float), ('sim_id', int)])
H0['sim_id'] = range(n_sim)

libE_specs = {'nworkers': nworkers,
    'comms': 'local'}
sim_specs = {'sim_f': sim_f, 'in':
    ['x'], 'out': [('y', float),
    ('calc_status', str)]}
gen_specs = {}
alloc_specs = {'alloc_f': alloc_f,
    'out': [('x', float)]}

exit_criteria = {'sim_max': len(H0)}

H, persis_info, flag = libE(sim_specs,
    gen_specs, exit_criteria,
    alloc_specs=alloc_specs,
    libE_specs=libE_specs, H0=H0)

```

Fig. 11: Pseudocode of the `libEnsemble` calling script.

6.1 Catalyst pipeline QoI extraction

The `Catalyst` library is successfully interfaced with PHASTA via `SENSEI` to enable real-time QoI extraction. First, the PHASTA adaptor file is edited to include additional fields that facilitate QoI computation. This adaptor file determines the fields that are passed from the PHASTA solver to `ParaView` during a simulation. Next, PHASTA is built with `SENSEI` enabled and the updated adaptor file. Finally, a `ParaView` pipeline is saved to a Python co-processing script that is read during the PHASTA run at a specified frequency. The execution of the `Catalyst` co-processing step results in some computational overhead which we assess via a strong scalability study.

```

# Setup distribution with stochastic
# dimension d variables
distribution = cp.Lid
              (cp.Uniform(-1,1),d)

# Construct PC expansion
polynomial_order = 4
polynomial_expansion = cp.orth_ttr
                    (polynomial_order , distribution)

# Evaluate expansion at samples to
#construct psi
psi = polynomial_expansion
    (*samples).T

# Solve for pc coefficients via least
# squares or compressed sensing
sigma = 0.001 #spgll solver tolerance
c_hat , resid , grad , info =
    spg.spg_bpdn(psi , evaluations ,
                sigma)

# Calculate Sobol indices from PC
# expansion
model_approximation = np.sum
                    (c_hat_spg*polynomial_expansion)

# First order sobol indice
s_l = cp.Sens_m(model_approximation ,
                distribution)
# Total sobol indice
s_t = cp.Sens_t(model_approximation ,
                distribution)
s_interactions = s_t-s_l

```

Fig. 12: Pseudocode of the chaospy sensitivity analysis. Samples is an array of samples $\{\xi_{ij}\}_{i=1}^N$, and evaluations are the corresponding simulated QoI \mathbf{u} from equation (4).

We performed a strong scalability study on the Argonne Leadership Computing Facility’s Theta computational resource with a $\sim 2572\text{K}$ element mesh of the diffuser, partitioned to run on 6, 12, 24 and 48 cores, with 1 MPI process per core. Theta is a 11.69-petaflops Intel-Cray supercomputer. Each simulation computes 25 time steps, and saves output data at 2 step intervals. In Figure 13 we plot the scalability study timing results for no write, conventional write and Catalyst write cases. The results are computed as the average of three repetitions. No write denotes a PHASTA run that neither saves data to restarts or via the Catalyst pipeline, conventional write indicates that a restart file containing the entire flow solution is written every 2 time steps and Catalyst write employs the Catalyst QoI pipeline to efficiently extract data every 2 time steps.

In Figure 13 (a) we observe that the performance of the varying write methods and the optimal scaling curve in terms of the wall-clock time are indistinguishable. In Figure 13 (b)

we plot the strong scalability factor, defined as

$$sf = \frac{t_{ref} \times nc_{ref}}{t \times nc}, \quad (7)$$

where t and nc are a given execution time and number of cores, and t_{ref} and nc_{ref} are the execution time and number of cores for the 6 core, no write, reference case. Optimal scaling would be a horizontal line with a scaling factor of 1. We observe in Figure 13 (a) that for all cases the parallel overhead increases with number of cores. Importantly, while Catalyst write is marginally less efficient than not writing at all, its performance is comparable to the conventional write method.

In the traditional workflow approach, where the QoI calculation is performed as a post-processing step, the entire solution field must be saved at a given time step to a restart file of approximately 205 Mb. In the Catalyst write approach, the file containing the QoI is just 4 Kb, indicating a storage reduction of approximately fifty thousand times. These results demonstrate that the significant memory advantages of writing output data via the Catalyst pipeline come at minimal computational overhead.

6.2 Ensemble generation

We have enabled the straightforward evaluation of ensembles of PHASTA simulations through the implementation of the `libEnsemble` Python library described in Section 5.2 [25]. `libEnsemble` facilitates two levels of parallelism. In the first, an individual PHASTA simulation may employ numerous processors, in this demonstration we partition a 241k element mesh and conduct each simulation with 24 processors. In the second, `libEnsemble` manages individual PHASTA evaluations and can launch new simulations immediately after previous simulations are finalized and resource becomes available. The user selects or provides functions to generate simulation input, carry out a simulation, and manage the ensemble of simulations. This setup allows the user to tailor `libEnsemble` to their specific solver requirements. For instance, to obtain an ensemble of PHASTA simulations we create a custom simulation script that for a given input realization creates a run directory with the necessary geometry and restart files, reads the inputs from file, writes the inputs to the `solver.inp` script and executes PHASTA.

We have developed and documented a set of PHASTA specific `libEnsemble` scripts and that enable the parallel evaluation of ensemble members, allowing much greater flexibility in job submission. `libEnsemble` with PHASTA has been implemented on local resources, and additionally on the Argonne Leadership Computing Facility’s Cooley cluster. Cooley has 126 compute nodes, each with 12 cores and 1 NVIDIA Tesla K80 GPU.

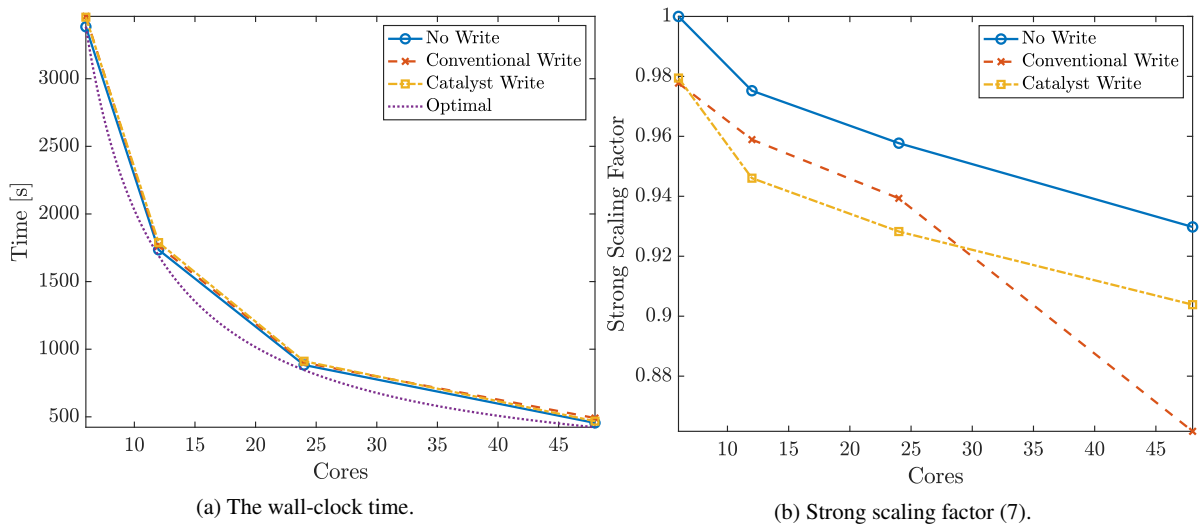


Fig. 13: Strong scalability study comparing the performance of no write, where no output data is saved, conventional write, where output data is written to restart files and Catalyst write where PHASTA is equipped with the Catalyst pipeline for efficient QoI extraction. In each case we plot the average of three repetitions of a 25 time step simulation with output data saved at every second step. 13 (b) presents the corresponding strong scalability factor

6.3 Global sensitivity analysis

We apply global sensitivity analysis via PC expansions, detailed in Section 5.1.2, to the 2D aggressive subsonic diffuser. Our objective is to understand the influence of the UB and LB velocities and the UB position on the length of the recirculation area (1). While the UB and LB provide us with flow control, the mass flow rate diverted to these blowers increases intake drag of the entire diffuser. Therefore, we want to determine a blower configuration that reduces the recirculation area while expending minimal diverted flow.

To explore the design space we treat the UB and LB velocities and amplitudes and the UB position as uniform random variables. The variables and their respective intervals are summarized in Table 1. The limits were determined through computational steering via *in situ* exploration of the parameter space. Figure 14 shows the PC based Sobol indices constructed from 495 samples using polynomial order $p = 4$. The PC validation error was 10.1%. In Figure 14 we see that the most influential parameter is the UB velocity mean, while the LB velocity amplitude has minimal effect on the QoI variance. The UB position φ also plays a significant role. Further, it is apparent that the recirculation length is sensitive to the interactions between all the parameters. This matches our expectation given the tug of war dynamic between the upper and lower blower's over the core flow.

We plot in Figure 15 the entire ensemble in terms of mass cost and recirculation length, colored by the sampled input parameters. We exclude A_L (see Table 1) from Figure 15 due to its small sensitivity found in Figure 14. Mass cost is computed simply as the sum of the UB and LB mean

Parameter	Symbol	Distribution
UB velocity mean	μ_U	$U[1.43, 2.85] \%$
UB velocity amplitude	A_U	$U[0.00, 2.13] \%$
LB velocity mean	μ_L	$U[0.00, 2.34] \%$
LB velocity amplitude	A_L	$U[0.00, 1.17] \%$
UB Position	φ	$U[0, 1]$

Table 1: Upper blower (UB) and lower blower (LB) input parameters. Velocity metrics are given as a percentage of the total diffuser mass flow rate.

velocities. Consequently, it is no surprise that in Figures 15 (a) and (c) we find that increasing the mean mass flow rates is correlated with higher mass cost. However, this increase in mass is correlated with our aim of reducing the recirculation length. The best performing samples are those that produce the smallest recirculation length while expending the least mass. In general, these samples correspond to higher blower positions as illustrated in Figure 15 (d). While Figure 15 (d) implies that moving the blower position even further downstream may achieve better results, we refrain from doing this given the severe simplifications of this study, namely incompressible and 2D flow. Future work could transition examination of the blower position into 3D and compressible flow.

We select a sample, identified by a red circle in Figure 15, as flow configuration that yields low recirculation length and minimizes mass cost. This configuration reduces the recirculation length to 0, at a mass cost of approximately 4.5% the core mass flow. In Figure 16 we compare our chosen flow configuration to the diffuser no active flow control. The

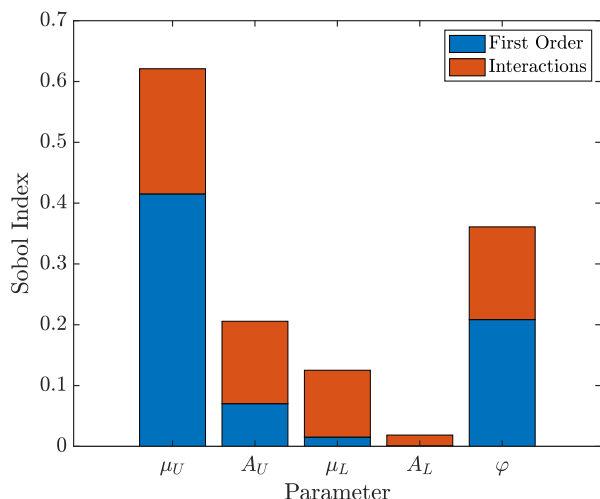


Fig. 14: Global sensitivity analysis of the 2D aggressive subsonic diffuser. This study was carried out in Python with `chaospy` and an order $p = 4$ PC expansion from 495 samples with validation error *error*. The sum of the interactions and first order is the total order.

time averaged stream-wise velocity is plotted. We observe in Figure 16 (a) that the recirculation area, outlined in black, is very large. In Figure 16 (b) the core flow has been bent upwards by the action of the UB and the recirculation area is reduced to 0.

7 Conclusions

In this work we introduce an immersive simulation software framework that enables practitioners to evade many of the typical workflow inefficiencies in solving systems of PDEs. We describe the *in situ* interface SENSEI and infrastructure Catalyst and provide software linkages to show their interaction with the PDE solver. In addition we demonstrate the use of complementary ensemble simulation software tools `libEnsemble`, for intuitive evaluation of simulation ensembles, and `chaospy`, for straightforward implementation of UQ techniques.

We demonstrate the efficacy of this immersive simulation framework on a 2D diffuser problem in the PHASTA. With SENSEI and Catalyst we implement computational steering on the diffuser problem, including parametric geometry deformation, that allows for rapid design space exploration and eliminates the need of writing data to disk. We show that using a Catalyst pipeline to save output QoI requires minimal computational overhead, scales well, and reduces memory demands by approximately fifty thousand times.

A global sensitivity analysis integrates all the immersive simulation components together. Computational steer-

ing allows the practitioner to test input parameters' influence, guide the selection of appropriate parameter limits and determine a useful QoI. The ensemble evaluation with `libEnsemble` is fast, intuitive, and takes full advantage of massively parallel resources. Moreover, the saving of output QoI with pre-constructed Catalyst pipeline drastically reduces the total memory footprint. Finally, we use `chaospy` to calculate Sobol' indices and determine the optimal diffuser configuration.

Our future research includes a plethora of advancements to immersive simulation capabilities. While the application of UQ techniques as a post-processing analysis is useful, the ability to establish sample limits during computational steering and launch ensembles interactively from within Paraview GUI would lead to yet more rapid insight extraction. In this work we provide a proof of concept for parametric geometry deformation that is designed specifically for this problem's geometry. More general geometric deformation tools are an interesting line of inquiry.

Acknowledgments

The work was supported by National Science Foundation grant 1740330 (SSE) and Department of Energy grant DE-SC0021411 (FASTMath SciDAC).

The SENSEI enhancements developed by Kitware were supported by the Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under a subcontract from Lawrence Berkely National Laboratory No. 7566608, through the grant "Scalable Analysis Methods and In Situ Infrastructure for Extreme Scale Knowledge Discovery."

References

1. Ahrens J, Geveci B, Law C (2005) Paraview: An end-user tool for large data visualization. The visualization handbook 717
2. ANSYS (2016) Ansys fluent - cfd software — ansys. <http://www.ansys.com/products/fluids/ansys-fluent>
3. Asghar A, Stowe RA, Allan WD, Alexander D (2017) Entrance aspect ratio effect on s-duct inlet performance at high-subsonic flow. *Journal of Engineering for Gas Turbines and Power* 139(5)
4. Ayachit U, Bauer A, Geveci B, O'Leary P, Moreland K, Fabian N, Mauldin J (2015) Paraview catalyst: Enabling in situ data analysis and visualization. In: *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pp 25–29
5. Ayachit U, Bauer A, Duque EP, Eisenhauer G, Ferrer N, Gu J, Jansen KE, Loring B, Lukic Z, Menon

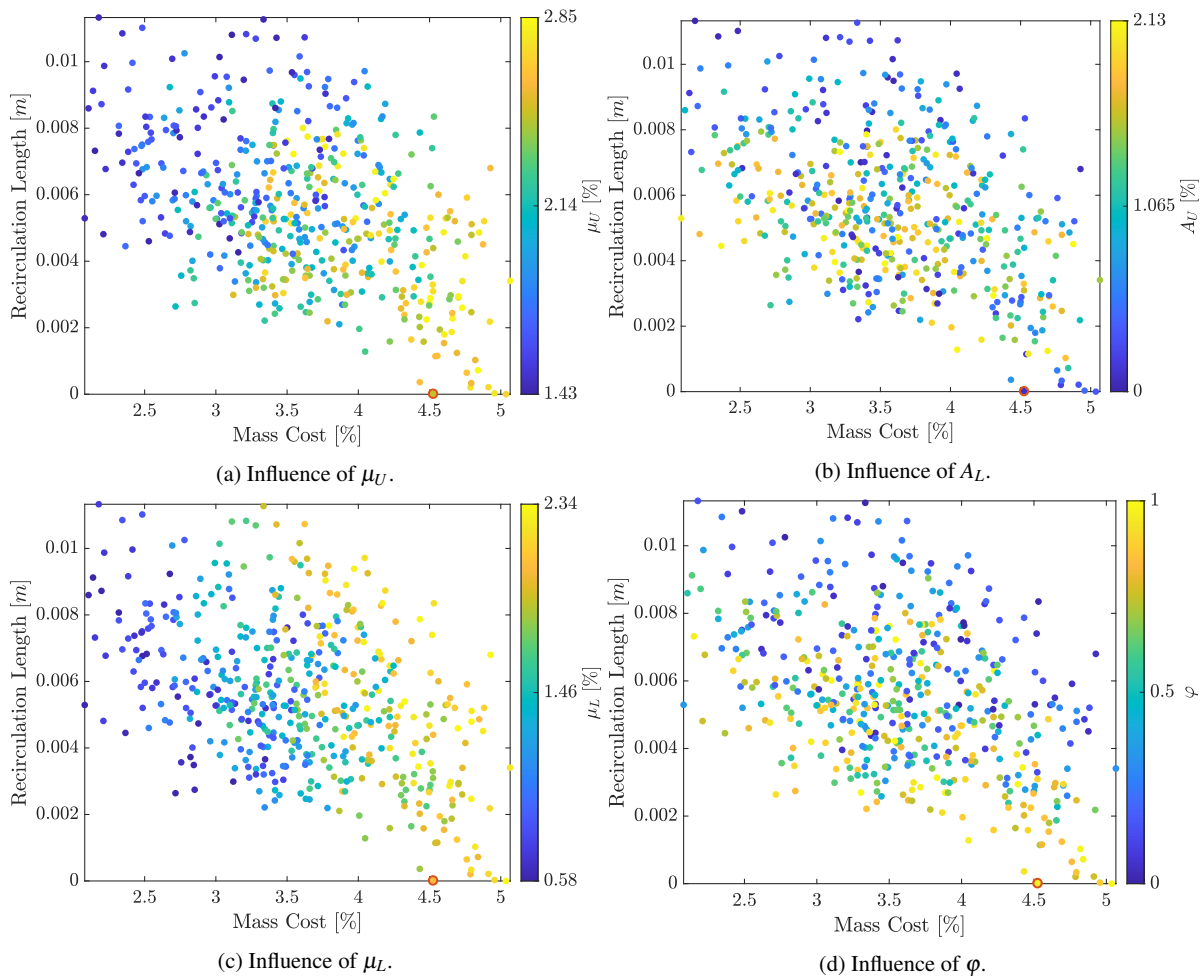


Fig. 15: Ensemble members in terms of mass cost and recirculation length colored by different input parameters. The chosen optimal flow configuration is circled in red.

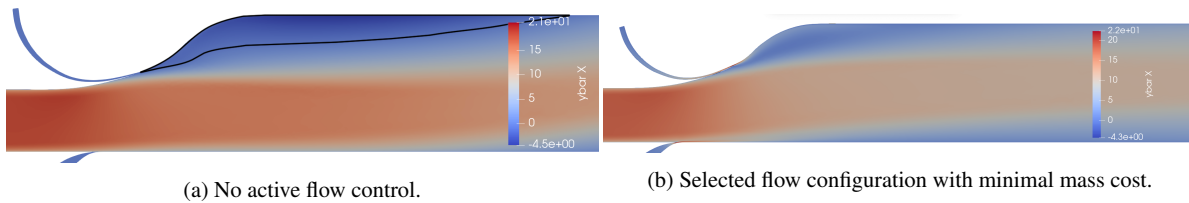


Fig. 16: Influence of active flow control on the recirculating region showing (a) no flow control and (b) the best performing configuration found via an ensemble of simulations. The recirculation region is outlined in black.

- S, et al. (2016) Performance analysis, design considerations, and applications of extreme-scale in situ infrastructures. In: SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, pp 921–932
6. Ayachit U, Whitlock B, Wolf M, Loring B, Geveci B, Lonie D, Bethel EW (2016) The sensei generic in situ interface. In: 2016 Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV), IEEE, pp 40–44
 7. Bauer AC, Abbasi H, Ahrens J, Childs H, Geveci B, Klasky S, Moreland K, O'Leary P, Vishwanath V, Whitlock B, et al. (2016) In situ methods, infrastructures, and applications on high performance computing platforms. In: Computer Graphics Forum, Wiley Online Library, vol 35, pp 577–597
 8. Buice CU, Eaton JK (1996) Experimental investigation of flow through an asymmetric plane diffuser. CTR Annual research briefs 1996:243–248

9. Camata JJ, Silva V, Valdúriez P, Mattoso M, Coutinho AL (2018) In situ visualization and data analysis for turbidity currents simulation. *Computers & Geosciences* 110:23–31
10. Chang TH, Larson J, Watson LT, Lux TC (2020) Managing computationally expensive blackbox multiobjective optimization problems with libensemble. In: 2020 Spring Simulation Conference (SpringSim), IEEE, pp 1–12
11. Childs H, Brugger E, Whitlock B, Meredith J, Ahern S, Pugmire D, Biagas K, Miller M, Harrison C, Weber GH, Krishnan H, Fogal T, Sanderson A, Garth C, Bethel EW, Camp D, Rübel O, Durant M, Favre JM, Navrátil P (2012) Visit: An end-user tool for visualizing and analyzing very large data. In: *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, pp 357–372
12. Constantine P, Howard R, Glaws A, Grey Z, Diaz P, Fletcher L (2016) Python active-subspaces utility library. *Journal of Open Source Software* 1(5):79
13. Diaz P, Doostan A, Hampton J (2018) Sparse polynomial chaos expansions via compressed sensing and d-optimal design. *Computer Methods in Applied Mechanics and Engineering* 336:640–666
14. Doostan A, Owhadi H (2011) A non-adapted sparse approximation of pdes with stochastic inputs. *Journal of Computational Physics* 230(8):3015–3034
15. Fabian N, Moreland K, Thompson D, Bauer AC, Marion P, Gevecik B, Rasquin M, Jansen KE (2011) The paraview coprocessing library: A scalable, general purpose in situ visualization library. In: 2011 IEEE Symposium on Large Data Analysis and Visualization, IEEE, pp 89–96
16. Feinberg J, Langtangen HP (2015) Chaospy: An open source tool for designing methods of uncertainty quantification. *Journal of Computational Science* 11:46–57
17. Foster I, Ainsworth M, Allen B, Bessac J, Cappello F, Choi JY, Constantinescu E, Davis PE, Di S, Di W, et al. (2017) Computing just what you need: Online data analysis and reduction at extreme scales. In: *European conference on parallel processing*, Springer, pp 3–19
18. Fu J, Liu N, Sahni O, Jansen KE, Shephard MS, Carothers CD (2010) Scalable parallel i/o alternatives for massively parallel partitioned solver systems. In: 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), IEEE, pp 1–8
19. Ghanem RG, Spanos PD (1991) Stochastic finite element method: Response statistics. In: *Stochastic Finite Elements: A Spectral Approach*, Springer, pp 101–119
20. Hadigol M, Doostan A (2018) Least squares polynomial chaos expansion: A review of sampling strategies. *Computer Methods in Applied Mechanics and Engineering* 332:382–407
21. Haimes R, Dannenhoffer J (2013) The engineering sketch pad: A solid-modeling, feature-based, web-enabled system for building parametric geometry. In: 21st AIAA Computational Fluid Dynamics Conference, p 3073
22. Haimes R, Jordan K (2001) A tractable approach to understanding the results from large-scale 3d transient simulations. In: 39th Aerospace Sciences Meeting and Exhibit, p 918
23. Hampton J, Doostan A (2015) Coherence motivated sampling and convergence analysis of least squares polynomial chaos regression. *Computer Methods in Applied Mechanics and Engineering* 290:73–97
24. Hampton J, Doostan A (2015) Compressive sampling of polynomial chaos expansions: Convergence analysis and sampling strategies. *Journal of Computational Physics* 280:363–386
25. Hudson S, Larson J, Wild SM, Bindel D, Navarro JL (2019) libEnsemble users manual. Tech. Rep. Revision 0.5.2, Argonne National Laboratory, URL <https://buildmedia.readthedocs.org/media/pdf/libensemble/latest/libensemble.pdf>
26. Kuhlen T, Pajarola R, Zhou K (2011) Parallel in situ coupling of simulation with a fully featured visualization system. In: *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization (EGPGV)*, Eurographics Association Aire-la-Ville, Switzerland, vol 10, pp 101–109
27. Larsen M, Ahrens J, Ayachit U, Brugger E, Childs H, Geveci B, Harrison C (2017) The alpine in situ infrastructure: Ascending from the ashes of strawman. In: *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pp 42–46
28. Le Maître O, Knio OM (2010) Spectral methods for uncertainty quantification: with applications to computational fluid dynamics. Springer Science & Business Media
29. Li S, Marsaglia N, Garth C, Woodring J, Clyne J, Childs H (2018) Data reduction techniques for simulation, visualization and data analysis. In: *Computer Graphics Forum*, Wiley Online Library, vol 37, pp 422–447
30. Liu N, Fu J, Carothers CD, Sahni O, Jansen KE, Shephard MS (2010) Massively parallel i/o for partitioned solver systems. *Parallel processing letters* 20(04):377–395
31. Lofstead JF, Klasky S, Schwan K, Podhorszki N, Jin C (2008) Flexible io and integration for scientific codes through the adaptable io system (adios). In: *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pp 15–24

32. Longley J, Greitzer E (1992) Inlet distortion effects in aircraft propulsion system integration. AGARD Lecture Series 183
33. Mu D, Moran J, Zhou H, Cui Y, Hawkins R, Tatineni M, Campbell S (2019) In-situ analysis and visualization of earthquake simulation. In: Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning), pp 1–5
34. Obi S, Aoki K, Masuda S (1993) Experimental and computational study of turbulent separating flow in an asymmetric plane diffuser. In: Ninth Symposium on Turbulent Shear Flows, vol 305
35. Peng J, Hampton J, Doostan A (2014) A weighted ℓ_1 -minimization approach for sparse polynomial chaos expansions. *Journal of Computational Physics* 267:92–111
36. Peterka T, Bard D, Bennett JC, Bethel EW, Oldfield RA, Pouchard L, Sweeney C, Wolf M (2020) Priority research directions for in situ data management: Enabling scientific discovery from diverse data sources. *The International Journal of High Performance Computing Applications* 34(4):409–427
37. Pintas JT, de Oliveira D, Ocaña KA, Ogasawara E, Mattoso M (2013) Scilightning: a cloud provenance-based event notification for parallel workflows. In: International Conference on Service-Oriented Computing, Springer, pp 352–365
38. Rasquin M, Smith C, Chitale K, Seol S, Matthews B, Martin J, Sahni O, Loy R, Shephard MS, Jansen KE (2014) Scalable fully implicit finite element flow solver with application to high-fidelity flow control simulations on a realistic wing design. *Computing in Science and Engineering* 16(6):13–21
39. Santos E, Tierny J, Khan A, Grimm B, Lins L, Freire J, Pascucci V, Silva CT, Klasky S, Barreto R, et al. (2009) Enabling advanced visualization tools in a web-based simulation monitoring system. In: 2009 Fifth IEEE International Conference on e-Science, IEEE, pp 358–365
40. Schroeder WJ, Lorenson B, Martin K (2004) The visualization toolkit: an object-oriented approach to 3D graphics. Kitware
41. Sederberg TW, Parry SR (1986) Free-form deformation of solid geometric models. In: Proceedings of the 13th annual conference on Computer graphics and interactive techniques, pp 151–160
42. Sudret B (2008) Global sensitivity analysis using polynomial chaos expansions. *Reliability engineering & system safety* 93(7):964–979
43. Tannahill J, Lucas DD, Domyancic D, Brandon S, Klein R (2011) Data intensive uncertainty quantification: applications to climate modeling. In: Proceedings of the 2011 High Performance Computing Networking, Storage and Analysis Conference
44. Tchoua R, Klasky S, Podhorszki N, Grimm B, Khan A, Santos E, Silva C, Mouallem P, Vouk M (2010) Collaborative monitoring and analysis for simulation scientists. In: 2010 International Symposium on Collaborative Technologies and Systems, IEEE, pp 235–244
45. Tennøe S, Halnes G, Einevoll GT (2018) Uncertainty: A python toolbox for uncertainty quantification and sensitivity analysis in computational neuroscience. *Frontiers in neuroinformatics* 12:49
46. Thompson D, Fabian ND, Moreland KD, Ice LG (2009) Design issues for performing in situ analysis of simulation data. Technical Report SAND2009-2014, Sandia National Laboratories pp 7–18
47. Vishwanath V, Hereld M, Morozov V, Papka ME (2011) Topology-aware data movement and staging for i/o acceleration on blue gene/p supercomputing systems. In: SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, pp 1–11
48. Wang C, Duan Q, Tong CH, Di Z, Gong W (2016) A gui platform for uncertainty quantification of complex dynamical models. *Environmental Modelling & Software* 76:1–12
49. Wetterer-Nelson C, Jansen KE, Evans JA (2021) Interactive geometry modification of high performance finite element simulations. in progress
50. Whiting CH, Jansen KE (2001) A stabilized finite element method for the incompressible Navier-Stokes equations using a hierarchical basis. *International Journal for Numerical Methods in Fluids* 35(1):93–116
51. Whitlock BJ, Laganella C, Duque EP (2021) Scoreboard: Management and creation of in situ and in transit data extractions via computational steering. In: AIAA Scitech 2021 Forum, p 1599
52. Xiu D (2010) Numerical methods for stochastic computations: a spectral method approach. Princeton university press
53. Xiu D, Karniadakis GE (2002) The wiener–askey polynomial chaos for stochastic differential equations. *SIAM Journal on Scientific Computing* 24(2):619–644
54. Yi H, Rasquin M, Fang J, Bolotnov IA (2014) In-situ visualization and computational steering for large-scale simulation of turbulent flows in complex geometries. In: 2014 IEEE International Conference on Big Data (Big Data), IEEE, pp 567–572