juliacon

Julia for HPC: In Situ Data Analysis with Julia for Climate Simulations at Large Scale

Li Tang[1], Soumya Dutta[1, 2], Natalie Klein[1], Wayne Yu Wang[3], Jonathan David Wolfe[1], Luke Van Roekel[1], Nathan Urban[4], Ayan Biswas[1], and Earl Lawrence[1]

[1]Los Alamos National Laboratory
[2]Indian Institute of Technology Kanpur
[3]University of Michigan
[4]Brookhaven National Laboratory

ABSTRACT

Fast-evolving data science techniques have enabled scientific discoveries by providing the ability to analyze large amounts of scientific data and extract relevant patterns. However, the ever-increasing gap between computing speed, which continues to increase, and data input/output (I/O) bandwidth, which remains relatively constant, often prevents full utilization of the data; we are likely unable to save all of the generated data for analysis. In situ data analysis techniques seek to address this issue by completing analysis and pattern extraction from generated scientific data while the application is running, and the in situ data analysis only involves compute and in-memory data I/O, which improves performance. In this paper, we develop an infrastructure for coupling a popular high-level data science programming language, Julia, with the large-scale production-level climate code Energy Exascale Earth System Model (E3SM). To demonstrate the infrastructure, we develop two in situ data analysis methods in Julia and evaluate their performance and the infrastructure overhead. Our results show that our in situ Julia data analysis methods are able to detect extreme weather events and characterize climate patterns with insignificant infrastructure overhead. Furthermore, our infrastructure allows user-friendly development and deployment of new Julia data analysis modules without the need to recompile the simulation code, giving data analysts a simple new tool for in situ analysis.

Keywords

Julia, In situ, High-performance computing, Climate simulations

1. Introduction

Computing devices are evolving much faster than storage devices because of their different ratios of performance improvement to Research and Development (R&D) investment in the post-Moore's Law era [34]. This trend enlarges the performance gap between our computing and storage devices and greatly limits our ability to analyze all the generated scientific simulation data. When run on High-Performance Computing (HPC) systems, modern physics simulations employ thousands of nodes, generating large amounts of data. For example, at a high resolution, the three-dimensional physics model state alone can be tens of gigabytes *per* time slice. While this data could possibly be compressed and then stored with high-performance input/output (I/O) for post-processing [24, 32], data movement and post-processing time can be extremely long. To address these issues, we adopt in situ data analysis to analyze physics simulation data as the simulation is running. The in situ data analysis design involves several design and performance trade-offs in terms of coupling the physics simulation and data analysis, and we will explore some of the trade-offs in this paper.

Our target HPC application is Energy Exascale Earth System Model (E3SM) [18]. E3SM is the Department of Energy (DOE) state-of-the-science earth system model, and we focus on E3SM's atmosphere component (EAM) in this study. Developing data science techniques is extremely costly by using HPC programming languages such as C/C++ and Fortran due to the ecosystem lacking, so most data analysis programs are implemented by using Python. However, Python's performance is usually slow on HPC systems and we decided to use Julia as the programming language of developing data analysis modules because of its superior performance for data analysis and rich support of data analysis packages [23]. Therefore, the major challenge of employing in situ data analysis in legacy scientific applications is that modern advanced data science techniques are usually not implemented in the same way as the HPC scientific codes. In this paper, we hypothesize that the Fortran-based EAM can be effectively coupled with in situ Julia-based data analysis modules on HPC systems.

To validate our hypothesis and demonstrate the in situ Julia data analysis infrastructure, we implement two in situ data analysis methods in Julia: (1) detection of Sudden Stratospheric Warming (SSW) with generalized extreme value (GEV) analysis, and (2) characterization of large-scale climate patterns via principal components analysis (PCA). SSW events are extreme events in the upper atmosphere (approximately 20km above ground level) that can cause the temperature to rise as much as 50^{o}C in only a few days. This extreme warming can destabilize the polar vortex, which can in turn lead to rapid swings in surface air temperature. To identify SSWs, we developed an in situ analysis method that documents every occurrence when the zonal mean of the zonal wind becomes reversed (easterly) at 60°N and 10 hPa and lasts for at least 10 consecutive days in the northern hemisphere winter. To characterize surface temperature changes that may result from SSWs, we fitted in situ generalized extreme value (GEV) models to the daily minimum surface temperatures at each spatial point in the continen-

tal United States (CONUS) via streaming variational inference [6]. By updating two separate sets of models (one following SSWs and one not), we examined differences in extreme temperature behavior. Our second data analysis method focuses on PCA of the surface temperatures across the globe. PCA is a common statistical technique for identifying patterns in data that best explain variation in the data. Our TributaryPCA method estimates the top spatial principal component patterns in a streaming, distributed fashion [36], whereas traditional analysis would need to save data from all spatial locations across many time steps to compute the principal components.

This paper presents a practical case study that uses Julia-based analysis in situ with a large-scale HPC application. The major benefit of this work is that it provides a reference design for coupling Julia with modern large-scale HPC applications which could be applied to other HPC applications with minimal effort. In addition, data scientists could develop new Julia data analysis methods and insert them into the interface without recompiling the HPC code. Another benefit of this work is an accurate and quantitative view of how the Julia infrastructure and coupling impact the original HPC application, which helps provide actionable guidance to HPC users before coupling Julia with their target HPC applications. To help HPC users adopt the in situ approach and Julia to meet their needs, this work provides the following key contributions:

(1) We develop a novel in situ data analysis infrastructure for coupling E3SM with Julia for in situ data analysis at large-scale with high productivity.
(2) We develop two in situ Julia data analysis methods for E3SM's EAM data at large-scale for identifying sudden stratospheric warmings and identifying dominant spatial patterns in temperature.
(3) We evaluate the performance of this coupling infrastructure on a large-scale HPC system and study various design considerations, leading to guidelines for adopting Julia for in situ data analysis on other HPC applications.
(4) We have open-sourced the code and instructions for reproducing this in situ data analysis with E3SM in `https://github.com/ltang85/In-Situ-data-Analysis-with-Julia-for-Climate-Simulations-at-Large-Scale/tree/main/src`.

This paper is organized in the following structure. Section 2 provides the background information and discusses some related works. Section 3 presents the in situ infrastructure that couples the EAM and the in situ Julia modules. Section 4 describes the algorithms of our two in situ Julia modules. Section 5 shows the experimental results, and section 6 summarizes this paper.

2. Background and Related Work

In this section, we describe the E3SM HPC simulation, the Julia programming language, and the general concept of in situ data analysis before discussing recent closely related work.

Our work aims to provide data scientists with a performant interface for developing in situ data analysis algorithms without directly interacting with complex HPC codes or relying on post-hoc analysis. However, there are three challenges in doing so. First, E3SM's EAM is written in Fortran and adopts Message Passing Interface (MPI) for large-scale parallelism [35], and there are no Fortran APIs for embedding Julia into EAM. Second, one MPI rank of EAM is only able to access to its local data (e.g., velocity and temperature), and each in situ Julia data analysis instance can only access the same local data of its parent EAM MPI rank. Lastly, EAM is compiled and interacts with other E3SM components, and the in situ Julia data analysis should be noninvasive to other E3SM components. To address these challenges, we develop and implement an infrastructure for coupling the Julia runtime with E3SM, which enables effective transfer of data and consistent MPI communication between Julia and E3SM through a minimalist design.

2.1 E3SM

E3SM is the new earth system model developed by DOE to answer critical DOE mission questions relating to energy and water security *and* run efficiently on emerging HPC architectures. E3SM includes components to model the atmosphere, land, rivers, ocean, and sea ice. The atmosphere component, EAM, utilizes a spectral element discretization on a cubed sphere grid. The model uses approximately 110 km resolution in the horizontal and 72 vertical layers with variable spacing. The model top is at 60km above ground level. A detailed analysis of its performance is given in [40, 41, 30, 31, 18]. EAM has a number of key features relative to its predecessor, which are described in [18]. Of interest to our work is the linearized ozone photochemistry, which improves EAM simulations of the stratosphere, which is critically important for our SSW analysis.

2.2 Julia

Julia is a dynamically typed programming language that is for general-purpose application development and provides many Python-like features (e.g., rich package support) [5]. The first public Julia release was introduced in 2013 and it is designed to provide users with the speed of C/C++/Fortran [5] while giving a convenient high-level programming environment. Julia is a built on top of the modularized LLVM compiler infrastructure [27], and features a Just-In-Time (JIT) compilation engine [2]. Therefore, Julia is ideal for agile development of advanced data science algorithms due to its speed and effectiveness in solving computationally complex problems. Also, Julia has support for parallelism and concurrency, which are crucial for HPC applications. In this paper, our major goal is to increase productivity by allowing data scientists to focus on writing high-level, performant numerical algorithms for legacy HPC applications at large-scale.

2.3 In Situ Data Analysis

Conventionally, large-scale HPC applications write their simulated scientific data to slow disks for post-hoc usage such as data analysis and visualization. As discussed before, the speed gap between computing and I/O is increasing, and modern supercomputers can generate large amounts of data that may not be fully stored. Therefore, the in situ approach is a processing technique that conducts the conventional post-hoc data analysis on the fly while the HPC application is still running, rather than using the data after it has been dumped to disk. That is, conventional disk I/O is converted to in-memory processing, which increases speed and enables dynamical analysis on full, rather than compressed and stored, simulation data. Employing this in situ technique enables novel data analysis methods that require large amounts of high-resolution simulation data from E3SM. Two conventional alternatives are data compression and high-performance I/O. For example, the Software for Caching Output and Reads for Parallel I/O (SCORPIO) library is used by E3SM for parallel data I/O [32].

2.4 Related Work

The in situ approach has been studied in recent years and has been employed in different aspects of HPC. First, there are existing in situ frameworks, such as Ascent [26], ParaView Catalyst [14], SENSEI [33], VisIt libSIM [38], and SmartSim [9]. These developed frameworks aim to help users build data analysis and data visualization, but none of them supports the Julia programming language. Second, the open-sourced ADIOS/ADIOS2 [28, 17] data I/O system and the studies of Chimbuko/CODAR [25] have explored data reduction for in situ approaches and researched the performance optimization of the whole in situ data analysis workflow. Last, previous studies have been made on optimizing the complex workflow that involves complex HPC applications and post-hoc data analysis. However, these studies still require significant I/O on disk storage and are not able to help reduce the data. For a more comprehensive summary of in situ infrastructures, readers are referred to [4, 10].

The demand for scalable algorithms for analyzing climate data is growing rapidly as computational climate simulations scale up, leading to growing output data size and prompting scientists to turn to in situ analysis approaches. For example, in situ analysis of eddies in large-scale MPAS-Ocean simulation data was done by Woodring et al. [39]. In this work, we focus on two data analysis questions of interest to climate scientists and demonstrate their in situ application. For extreme weather event analysis, the connection of SSW with the polar vortex events is a focus area for climatologists [3]. More recently, scientists have also used extreme-value distribution models for capturing rare climate phenomena and further analyzing them in detail [22, 37]. In our recent work, the usefulness of in situ SSW analysis was also demonstrated [12]. For general summarization of large-scale patterns, principal components analysis (PCA) is a standard method that typically requires access to the full simulation data across time steps to capture seasonal trends and deviations. TributaryPCA is an adaptation of PCA, a statistical method for reducing the dimensionality of big data, to the in situ setting. Ordinary PCA summarizes variation by projecting high-dimensional data into a lower-dimensional space optimized to retain fundamental properties of the data). TributaryPCA is a distributed version of an existing algorithm for online PCA (Oja's algorithm) that uses the MPI standard for communication. In this way, TributaryPCA is able to estimate the principal components from data that arrives sequentially over time and that is distributed across compute nodes. In previous work from our team, the accuracy and computational efficiency of the TributaryPCA method was demonstrated [36], but it was not applied in situ. An open-source Julia implementation is available at `https://github.com/lanl/PRISM/tree/master/TributaryPCA.jl`.

3. In Situ Infrastructure

Our in situ infrastructure consists of a Fortran interface, a C interface, and a Julia interface, which we describe in turn after introducing the overall design.

3.1 Overall Design

As EAM is our target, the primary design consideration in coupling Julia with E3SM is the identification of an appropriate entry point in E3SM's EAM for calling in situ Julia data analysis modules. EAM is simulated in the time-step style, so we place the entry point of calling in situ data analysis in the control of data output for each time step. The second design consideration is the management of in situ Julia data analysis. We implement the Julia management

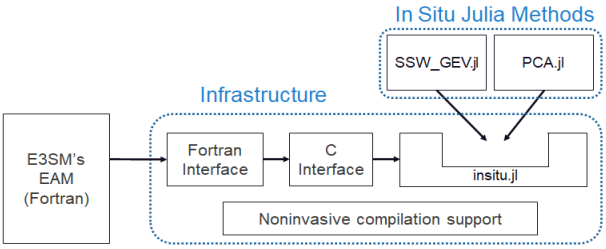

Fig. 1: Overall infrastructure design.

in EAM at the in situ entry point because E3SM schedules the jobs of EAM and other components. Last, due to the lack of APIs for embedding Julia in Fortran with MPI, we develop a C interface to help the communication between E3SM and Julia. The MPI communication relies on the MPI interface for the Julia language [7]. To make the C interface flexible, we also define and develop a simple Julia interface for quickly adopting various in situ Julia data analysis methods without interacting with E3SM. The whole design is shown in Figure 1. Our primary target data consumption pattern of the in situ data analysis module is that the in situ module can consume any intermediate simulation data in a non-invasive way, which is independent on the simulation output. Also, this in situ infrastructure enables superior data analysis capability over the alternative data compression or high-performance I/O solutions when the simulation is running at larger scales or higher resolutions.

3.2 Fortran Interface

To couple E3SM's EAM with Julia, we implement a Fortran-based in situ interface in EAM to manage in situ Julia data analysis and pre-process the EAM simulation data. Specifically, this interface is implemented in EAM's `stepon` component, but we note this interface could be used for in situ data analysis on other E3SM components with minimal effort. Two major tasks are implemented in our Fortran interface. First, this interface calls the worker function of actual in situ Julia data analysis at every time step, and the initialization/cleanup functions at the first/last time step. Second, the interface wraps the EAM simulation data with its supportive data (e.g., data length, time step) and passes the pointer of the wrapped data to the C interface for removing explicit data copy. Also, this adapter passes an internal E3SM MPI communicator to the C interface. When the E3SM simulation is running at large-scale, each MPI rank of EAM calls its own Fortran interface and initiates one Julia instance. For the time steps between the first and last time step, each in situ Julia instance returns the control back to its parent EAM after it finishes its data analysis. After Julia returns its control back to EAM, the Julia runtime is still working and keeps global variables alive for usage in the following time steps.

3.3 C Interface

The C interface stands between the Fortran and Julia interfaces, which aims to efficiently implement the functions in the Fortran interface for interacting with Julia. The C interface includes three major functions: initialization, cleanup, and worker, which creates an in situ Julia instance (by loading and initializing the in situ Julia module from a specified path), destroys the Julia instance, and passes the data from the Fortran interface to the in situ Julia instance (i.e., Julia interface). The following code snippet shows the major parts of the initialization and cleanup functions in the C interface.

Code 1: C interface initialization and cleanup.

```
jl_function_t *f;
void jl_init_(){
  jl_init__threading();
  jl_eval_string("Base.include(Main, \
  \"insitu.jl\")");
  jl_eval_string("using Main.mymodule");
  jl_module_t *mod_name = \
  (jl_module_t*)jl_eval_string("Main.mymodule");
  f = jl_get_function(mod_name, "streaming");
  return ;
}
void jl_exit_(){
  jl_atexit_hook(0);
  return ;
}
```

The worker function in the C interface aims to support efficient low-level data communication between E3SM and the in situ Julia data analysis. The key implementation of this C interface is shown in the following code snippet. Each MPI rank of EAM has access to only a local data block of EAM variables (e.g., velocity and temperature) and passes its local data block's pointer (e.g., `arr` and `p_arr` in the code snippet) to its own in situ Julia instance through the C interface. When the in situ Julia instance needs remote data (e.g., for computation of SSW) from other in situ Julia instances, `MPI.jl` is used in the Julia interface to implement the data communication between different in situ Julia instances. However, one key design challenge is to ensure that E3SM and Julia use the same MPI communicator for correct data communication, as current Julia C embeddings are not able to directly pass the MPI communicator. To address this challenge, we transfer the original Fortran MPI communicator to this C interface without conversions, and then wrap it as *jl_box_int32* and pass it to the Julia interface. The Fortran MPI communicator will be converted to Julia MPI communicator in the Julia interface. It is also important to root the Julia objects in this C interface between *JL_GC_PUSH3()* and *JL_GC_POP()*, preventing that the garbage collection (GC) frees the Julia references of `arr` and `p_arr` in this C interface accidentally. This is because `arr` and `p_arr` in the code snippet are allocated and deallocated in E3SM for every time step, and the Julia rooting can hold the references safely when running the Julia code.

Code 2: C interface worker.

```
void jl_worker_(double *arr, int* arrlen,  \
int* p_arr, int* p_len, MPI_Fint *Fcomm){
  jl_array_t *x = NULL, *y = NULL;
  jl_value_t *jl_comm = NULL;
  JL_GC_PUSH3(&x, &y, &jl_comm);
  x = jl_ptr_to_array_1d(jl_apply_array_type \
  ((jl_value_t*)jl_float64_type,1),arr,*arrlen,0);
  y = jl_ptr_to_array_1d(jl_apply_array_type \
  ((jl_value_t*)jl_int32_type,1),p_arr,*p_len,0);
  jl_comm = jl_box_int32(*Fcomm);
  jl_call3(f,(jl_value_t*)x,(jl_value_t*)y,jl_comm);
  JL_GC_POP();
  return ;
}
```

3.4 Julia Interface

The Julia interface mainly defines the format of input data, so all in situ Julia data analysis methods can be plugged-in without additional E3SM compilation, thanks to Julia's Just-in-time (JIT) compilation. As discussed previously, another key design of the Julia interface is to correctly convert the Fortran MPI communicator to Julia MPI communicator, which enables the algorithm developers to write their analysis routines using the same EAM MPI communicator. Due to the low-level MPI implementations, different MPI libraries (i.e., OpenMPI, MPICH, and MVAPICH) require specific converters; the following code snippet shows the converter implementations [15, 29, 20].

Code 3: Juila interface MPI conversion.

```
\\ OpenMPI
Jcomm = MPI.Comm(ccall((:MPI_Comm_f2c, \
        "openmpi/lib/libmpi.so"),Ptr{Cvoid}, \
        (Cint,),Cint(Jcomm)))

\\ MPICH and MVAPICH
Jcomm = MPI.Comm(Cint(Jcomm))
```

3.5 Noninvasive Compilation

The final design consideration is the compilation and execution design of coupling Julia with E3SM. The Fortran interface is implemented inside EAM, and the C interface file is placed in the same folder as the Fortran interface. The in situ Julia module is placed in the same run folder with the E3SM executable. As E3SM mixes the usage of GNU Make and CMake for combining and compiling different E3SM components, we have added the Julia compilation flags for the C and Fortran interfaces into the EAM CMake file (i.e., header files) and the top-level GNU Make file (i.e., Julia libraries). As we load the in situ Julia module via the Just-In-Time (JIT) scenario, the in situ Julia module is only compiled when it is called during runtime. Therefore, the whole infrastructure avoids recompiling the whole of E3SM when the in situ Julia module needs to be changed, and the in situ Julia module can be easily switched by only replacing the code of the same module file.

4. Data Analysis Modules

In this section, we describe the two demonstration data analysis modules: detecting SSW with extreme value modeling, and TributaryPCA.

4.1 Sudden Stratospheric Warming (SSW)

SSW events often lead to extreme cold temperatures across the United States, sometimes referred to as polar vortex events, which can have severe downstream impacts on energy systems (e.g., power grid) and on human lives. Accurate prediction of SSWs in simulations requires monitoring zonal winds at high temporal frequency; this is challenging in traditional simulation work flows due to data storage and access bottlenecks.

Our first in situ climate analysis algorithm detects extreme weather events called SSWs. By definition, SSW is characterized as a major midwinter warming that occurs when the daily zonal mean zonal winds at 60°N and 10 hPa (hectopascal) become easterly for at least 10 consecutive days between November and March [8, 1]. This event can result in extremely cold temperatures at the Earth's surface, causing hazardous weather and disrupting many socioeconomic sectors. Since the robust detection of rare SSW events requires spatiotemporal climate data at high temporal frequencies, for high-resolution EAM runs, post-hoc SSW detection capabilities are limited as storage of high-resolution and high-frequency

climate data would be prohibitive. Therefore, an in situ SSW detection algorithm is required.

Here, we briefly present the distributed SSW detection technique that we have developed and deployed in situ with EAM. During the simulation run, when a time marks the end of a day, we further process data from that time step. Initially, we compute the zonal mean zonal wind at each MPI process independently using zonal velocity (U-velocity). As the EAM mesh does not put grid points exactly at 60°N and 10 hPa, we first filter two layers of grid points, which are above and below the 10 hPA pressure level and fall within [59°N - 61°N] at each MPI process. Then, we linearly interpolate the zonal wind values to obtain values at 60°N and 10 hPa. Then, using an MPI reduction, we estimate the final global daily zonal mean zonal wind value. Based on the definition of SSW, this value needs to be negative for at least 10 consecutive days, so we maintain a global counter variable that keeps track of the number of consecutive days for which the value is negative. Once the value of that counter reaches 10, we record an SSW event for that simulation year (where the event continues until the mean zonal wind becomes non-negative again). A pseudo code for this algorithm is provided in Algorithm 1. Note that when an SSW event is detected, the climate scientists typically are interested in collecting more data from the following time steps, so the SSW detection can be used as a triggering event. Using SSW as a triggering event can reduce the overall in situ computation load since the SSW-specific analyses will only happen adaptively when an SSW event is detected.

Algorithm 1 In situ algorithm for SSW detection.

Input: EAM simulation data.
Output: A list of time steps and corresponding simulation years when SSW is detected.

```
for each EAM time_step do
    if (current_time_step == end_of_day) then
        Compute partial zonal mean zonal wind values
        at each MPI process.
        Compute the global zonal mean zonal wind
        values with MPI:Reduction.
        if (global zonal mean zonal wind < 0) then
            counter=counter+1
        else
            counter=0
        if ((counter == 10)) then
            Record current time step as the beginning of
            an SSW event.
    else
        continue
```

To characterize spatial patterns in surface temperature across CONUS following a detected SSW (compared to no detected SSW), we fit two separate GEV models, similar to [12]. These models represent the distribution of the daily minimum temperatures separately at each spatial location. Specifically, we transform the surface temperature data by subtracting 300°K and multiplying by negative one, then fit GEV models. Briefly, we fit Gumbel models with probability density function $p(x;\mu,\beta) = \frac{1}{\beta}\exp\left\{-\left[\frac{x-\mu}{\beta}+\exp\left(-\frac{x-\mu}{\beta}\right)\right]\right\}$ where the initial priors are $p(\mu) \sim \mathcal{N}(0,10)$ and $\beta \sim \text{LogNormal}(0,0,5)$. As described in [12], Algorithm 1, model parameters are updated periodically using a buffer of recent data, depending on whether or not an SSW event was recently detected; at each step, the approximate variational posterior distribution from the previous step is used as the prior when updating the model with the new data point to achieve approximate streaming Bayesian inference across time points. In this paper, we update the GEV models every three days, using the last three days of daily minimum temperatures as the data for modeling. The fitted GEV models then represent distributional information about the data through their parameter estimates, which are updated in situ without the need to save the raw data. In this work, each time the GEV models were to be updated, we used 10 Monte Carlo samples to evaluate each gradient and 500 total optimization iterations. We used the Adam optimizer from Turing [16] with a learning rate of 0.0001.

4.2 TributaryPCA

As described more fully in [36], TributaryPCA solves the standard PCA objective in a distributed, online fashion; we give a brief overview of the implementation here. Let $X \in \mathbb{R}^{d\times N}$ be a data set consisting of N samples with unknown covariance $\Sigma \in \mathbb{R}^{d\times d}$. In our application, N is the number of time points and d is the number of spatial locations. PCA seeks a k-dimensional orthonormal subspace $V \in \mathbb{R}^{d\times k}$ such that when the data are projected onto V, their variance is maximized. This amounts to solving

$$\max_{\substack{V\in\mathbb{R}^{d\times k}\\ V^TV=I_k}} \text{Tr}\Big(V^T\Sigma V\Big). \tag{1}$$

Equation (1) is maximized by the top-k eigenvectors of Σ, so the classical PCA calculation computes the top-k eigenvectors of the estimated covariance matrix. However, this approach is not feasible when data arrives sequentially (so that all N samples are not available at one time) or when the dimension of the data is large (so that it is not stored centrally on one compute node). In the streaming setting, the AdaOja algorithm [21] seeks to estimate the solution to Equation (1) using projected stochastic gradient descent with an automatically-tuned step size. That is, the estimated eigenvectors V are adjusted as each data point arrives (over time) using the gradient of Equation (1) evaluated at the new data point. This procedure consists of three key steps: a gradient computation, a learning rate update, and a QR decomposition (to ensure the orthogonality of the estimated eigenvectors). The TributaryPCA algorithm (1) computes the gradient in a distributed fashion using data stored on each node and MPI `allreduce`, (2) updates the learning rate in a distributed fashion using MPI `allreduce`, (3) applies the stochastic gradient update on each node, and (4) uses distributed linear algebra [11] to orthogonalize the estimated V (collected from all nodes using MPI `reduce`). In streaming PCA, data $X_i \in \mathbb{R}^{d\times 1}, i = 1, ..., N$ arrives sequentially and the goal is to compute V using all N samples, where V is updated sequentially with the current estimate denoted as V_i. The AdaOja algorithm relies on an initial learning rate α which we initialize to $1e-5$. In TributaryPCA, we have the additional complication that data X_i is partitioned across B compute nodes. The key portion of the in situ code (that would be called at each time step during the simulation) is shown in the following listing, where `X_par` is the data on the current node (of shape $d_b \times 1$ where d_B is the dimension of the data on a single compute node indexed by b), `V_par` is the node's spatial subset of the current global estimate of the eigenvectors (of shape $d_B \times k$), `comm` is the MPI communicator object, `master` denotes the root/master node, and `grad_par`, `abs2`, α are pre-initialized arrays. The code has three key steps: first, the gradient for the AdaOja streaming PCA update is computed; this quantity is $X_iX_i^TV_{i-1}$, which is computed with a

local dot product at each node followed by an `Allreduce` to combine results across nodes, and another multiplication locally. The second step is to update the learning rate based on the norm of the gradient and take a gradient step to update V_{i-1} to V_i based on data X_i. Finally, the result must be projected to ensure V_i is orthogonal. This is accomplished by QR decomposition $V_i = Q_i R_i$ where V_i is updated to the orthogonal Q_i. For distributed QR decomposition, we use CholeskyTSQR [11].

Code 4: Key algorithmic steps of TributaryPCA in Julia.

```
# Step 1: Compute gradient for AdaOja
XT_V_par = X_par' * V_par
XT_V = MPI.Allreduce(XT_V_par, +, comm)
mul!(grad_par, X_par, XT_V, 1.0/N, 0.0)
# Step 2: Update learning rate α
and apply gradient update
s = sum(abs2, grad_par, dims = 1)[:]
grad_norm_sq = MPI.Allreduce(s, +, comm)
α .+= grad_norm_sq
# Apply local gradient update
α_r = reshape(sqrt.(α), (1, size(α, 1)))
V_par .+= grad_par ./ α_r
# Step 3: Projection -- Orthogonalize to
#         form global update of V
VT_V_par = V_par' * V_par
VT_V = MPI.Reduce(VT_V_par, +, master, comm)
if node_id == master
    chol = cholesky!((VT_V + copy(VT_V')) / 2)
    Rinv = inv(chol.U)
else
    Rinv = nothing
end
Rinv = MPI.bcast(Rinv, master, comm)
V_par .= V_par * Rinv
```

5. Experiments

This section first summarizes the experimental setup for our in situ tests, including the application setup, conducted problems, build process and the underlying systems. Then we will show the in situ data analysis results of our in situ SSW and PCA Julia modules. Finally, we discuss the performance of this framework and some insights of conducting in situ Julia data analysis with large-scale scientific applications on modern supercomputers.

5.1 Setup

We have run all our experiments on a cluster, Grizzly, located at Los Alamos National Laboratory (LANL). Grizzly consists of 1,490 compute nodes, and each node is equipped with two Intel Broadwell Xeon E5-2695v4 2.1GHz processors (18 cores per CPU) and 128GB 2400 MHz DDR4 memory. Table 1 shows the Xeon E5-2695v4 specifications. The system runs on a Tri-Lab Operating System Stack (TOSS) and its nodes are connected by Intel Omni-Path Host Fabric Interface (HFI) single port via PCIex16. Grizzly uses Slurm for job scheduling and resource management. The total storage is 15PB.

To deploy our in situ algorithms, we have run the fully coupled E3SM simulation and accessed the data from its Atmosphere model, EAM, for analyses. The EAM module runs in conjunction with other coupled modules (MPAS-Ocean, MPAS Sea Ice, MPAS Land Ice, and Land Surface) to produce scientifically meaningful climate data. In this work, we use a realization of the Shared Socioeconomic Pathway (SSP) 585 scenario [13] with E3SM. This is an aggressive scenario that assumes the climate will experience

Table 1. : The E5-2695V4 CPU specifications

Specifications	Values
Total Cores	18
Total Threads	36
Max Turbo Frequency	3.30 GHz
Processor Base Frequency	2.10 GHz
Cache	45 MB
Memory Types	DDR4
Max Memory Bandwidth	76.8 GB/s

an increase in radiative forcing of 8.5 W/m^2. We use the standard E3SM V1 configuration with a 1° atmosphere and land (equivalent to 110 km at the equator), 0.5° river model (55 km), and an ocean and sea ice with mesh spacing varying between 60 km in the mid-latitudes and 30 km at the equator and poles [19].

Our tests fall into two data analysis tasks: SSW and PCA. Importantly, E3SM is only compiled once, and we can then switch between the PCA and SSW in situ Julia modules. The Julia modules are dynamically compiled and executed by using the Julia 1.8 version. Each in situ run is performed 10 times to mitigate system noise. We also add timers in the Slurm system for measuring performance and overhead.

5.2 In Situ SSW Detection and Characterization Results

After running for 365 simulation days (one simulated year), with the GEV models updated every three days, we found that the post-SSW GEV model was updated 11 total times while the non-SSW GEV model was updated 110 times. More detailed analysis of offline data, including the comparison of SSW and non-SSW models, can be found in [12]; for the purposes of this paper, we are interested in demonstrating the in situ, streaming modeling approach in general rather than drawing specific conclusions from the simulation in question. As a result, we will show the GEV modeling results for the non-SSW simulation periods because the models were updated more total times and therefore achieve lower parameter variance and greater interpretability. Figure 2 shows the GEV model parameters μ (location) and β (scale) for the non-SSW time periods at the conclusion of one simulated year. While the parameter values can be somewhat difficult to interpret directly, we emphasize that these saved parameter values, taking only a fraction of the storage space of the original data, can provide rich information about the distribution of minimum temperatures; for instance, we can use the model parameters to answer questions such as "What is the probability that the minimum temperature is more than 10 degrees Kelvin lower than the average value?" In this relatively short simulation run, we see that the spatial patterns in the parameter plots are somewhat rough and non-smooth, particularly for β; we expect that longer simulation runs and modifications to the GEV variational inference scheme could improve our estimates. However, it is promising that some noticeable spatial structure appears; for example, both μ and β appear to be smaller over the Atlantic Ocean, mirroring the tendency of temperatures to be more stable over the ocean (less fluctuation in the minimum value). We take these results as a promising sign that our models can capture relevant information, though due to the relatively short simulation time period, we leave making any claims based on these models for future work.

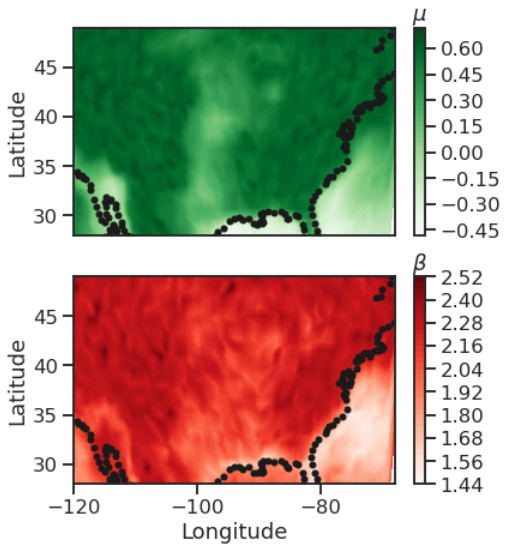

Fig. 2: Generalized extreme value (GEV) model parameter estimates at the end of one simulation year for the non-SSW time periods across the continental United States (coastline denoted with black points). Top: location parameter μ, which shows notable spatial patterns over the Atlantic Ocean and the Gulf of Mexico; in addition, a band of lower μ values appears to stretch through the Midwest region. Bottom: scale parameter β, which has less smooth spatial structure, but some similarities to the top figure in terms of lower values over bodies of water and through the Midwest region.

5.3 PCA Results

Over the one-year simulation run, we extracted the top three spatial principal components of the surface temperature variable using the TributaryPCA method; the principal components were updated in a streaming fashion at each simulation time step (6-hour time steps). Figure 3 shows spatial plots of these three principal components. Each component indicates a different spatial pattern of warming (red color) and cooling (blue color) extracted from the simulation data. In particular, the first component indicates a temperature differential between equatorial and polar regions; because the mean was not removed, this component can be interpreted as the mean pattern across the time steps. The second component indicates a temperature differential between much of the Eastern hemisphere and the Americas and parts of Africa. The third component indicates a temperature differential between Africa and Europe and other regions. It is plausible that the second and third components correspond to the regular diurnal cycle, which is likely one of the largest sources of regular variation in the one-year timespan considered here. While these components were extracted from a fairly short simulation run, the results demonstrate the ability of TributaryPCA to extract important climate patterns while running in a distributed in situ environment, without saving any raw data for post-processing.

5.4 Performance

This section studies the performance of our developed in situ framework and how the framework impacts the performance of E3SM. Based on these studies, we will make suggestions for integrating in situ Julia analysis with large-scale HPC applications.

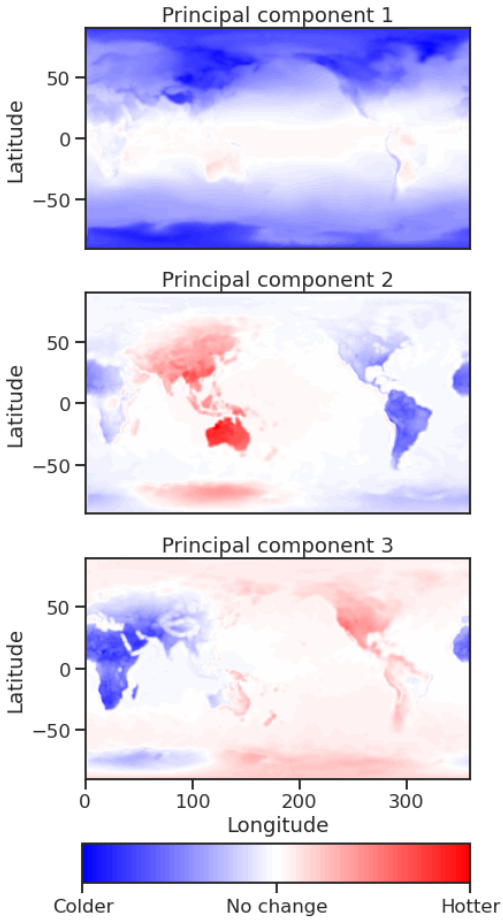

Fig. 3: First three spatial principal components obtained by in situ TributaryPCA method. Within each component, the white color indicates no change to surface temperature, while blue and red indicate temperatures deviating in different directions from zero. The first principal component appears to indicate the overall difference between the poles and the equator. The second principal component indicates differences between the Eastern hemisphere and Africa/the Americas, while the third component indicates differences in Africa and Europe compared to other regions.

5.4.1 Problem and Processing Element (PE) Layout.

The atmosphere component of E3SM, CAM, uses a spectral element dynamical core at 110-km resolution on a cubed sphere geometry. It has 72 layers with a top at approximately 60 km. The main atmosphere physics time step is 30 minutes and the ne30 grid used as the E3SM v1 main configuration has 5400 horizontal elements. So, we define nine different PE layouts in Table 2 to distribute the workload of 5400 atmospheric elements evenly (for performance evaluation), as well as the other components. Some of the other components run sequentially with CAM, using the same processors as it does, while other run concurrently on a different set of processors.

5.4.2 Framework Performance.

We first run the high-resolution E3SM problem with both of the SSW_GEV and PCA in situ analyses. For each analysis, we run

Table 2. : List of our PE Layouts. The PE layouts include three different numbers of MPI ranks and three different numbers of nodes.

PE Layout	# of nodes	# of PEs (MPI ranks)	# of PEs (MPI ranks) per node
100%Small	38	1350	36
50%Small	75	1350	18
25%Small	150	1350	9
100%Medium	50	1800	36
50%Medium	100	1800	18
25%Medium	200	1800	9
100%Large	75	2700	36
50%Large	150	2700	18
25%Large	300	2700	9

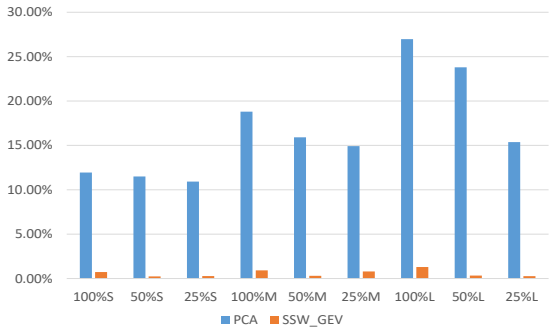

Fig. 4: In situ data analysis overhead (ratios of in situ data analysis time to the total EAM time).

with the predefined nine different PE layouts. Figures 4 and 5 show the in situ data analysis overhead and the infrastructure overhead, respectively. The overhead bars represent the ratios of the data analysis and infrastructure overhead time to the total EAM execution time. The overhead execution time includes the data preparation for the Julia module, loading Julia module, and cleaning up the Julia module. It can be seen from the results that our developed SSW_GEV only consumes less than 4% of the total EAM execution time. PCA is more computationally intensive than SSW_GEV, and it can take up to 26% of the total EAM execution time. This is because the main tasks of SSW_GEV and PCA are linear interpolation and Cholesky decomposition, respectively. The performance bottleneck of PCA is the expensive Cholesky decomposition. In addition, it can be observed that the in situ analysis consumes more time of the EAM execution time for larger layouts and higher PE usage on each node. This is because EAM generates less data per node in large layouts and the in situ analysis is less performance-sensitive than EAM in terms of data reduction. Also, higher PE usage per node raises memory pressure for the in situ analysis. We also observed that GC might potentially cause irregular system crashes when the in situ data analysis runs for a long time.

5.4.3 E3SM performance trends.

To study how the in situ analysis impacts the E3SM performance, we also run the original E3SM without the in situ framework. Figures 6 and 7 show the performance trends of the EAM components with different PE layouts for the SSW_GEV and PCA in situ experiments, respectively. As discussed in the previous section,

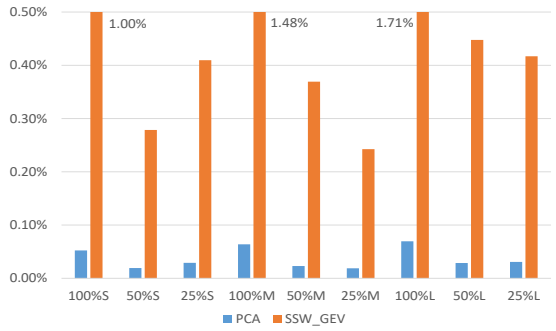

Fig. 5: In situ infrastructure overhead (ratios of in situ infrastructure overhead time to the total EAM time).

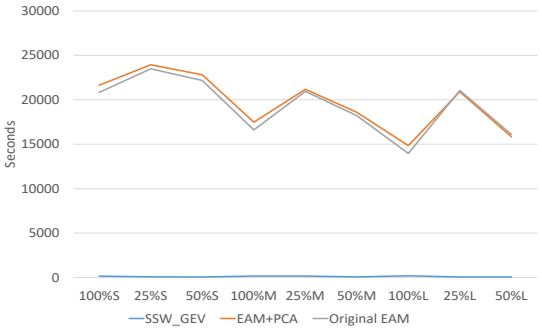

Fig. 6: Execution time trends of the SSW_GEV in situ analysis and the EAM component.

SSW_GEV is lightweight, and its performance trend is flat compared to the EAM component. Also, the EAM component itself is almost not impacted by the in situ framework. The only exception is the high PE usage per node and the reason is limited memory shared by E3SM and the Julia runtime and module. For PCA, the only performance overhead of the in situ framework is from the Julia analysis, where high PE usage per node can cause up to 6% additional overhead for SSW_GEV with the 100%L layout.

5.4.4 Performance variations across MPI ranks.

We examine the performance variations across all MPI ranks for SSW_GEV and PCA in this section. As we run the same high-resolution problem for 12 months for all of our experiments, the load balancing is only impacted the total number of PEs (i.e., S, M, and L). Figures 8 and 9 show the performance variation results for the SSW_GEV and PCA in situ experiments, respectively. The performance is represented by the processing rate, which is defined as the number of single-precision floating-point variables that can be processed in one second. The blue, orange, and grey bars represent the slowest, average, and fastest processing rates. First, it can be seen from the figures that SSW_GEV has higher variation than PCA, and this is due to the root PE of SSW_GEV being responsible for data collection. Second, medium PE usage (i.e., 50%) per

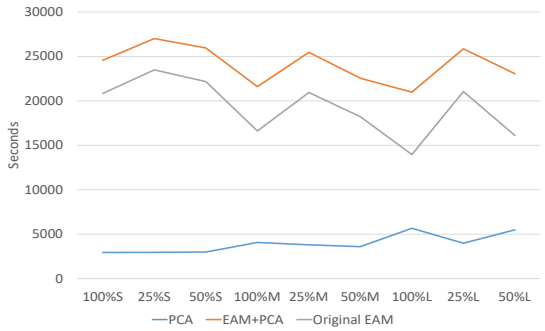

Fig. 7: Execution time trends of the PCA in situ analysis and the EAM component.

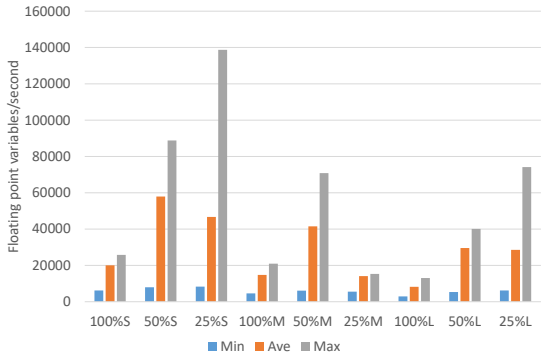

Fig. 8: Execution time of MPI ranks for the SSW_GEV in situ analysis.

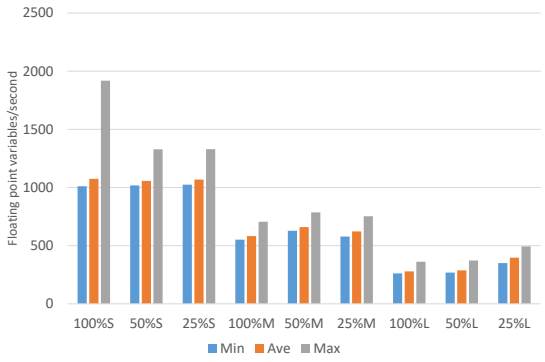

Fig. 9: Execution time of MPI ranks for the PCA in situ analysis.

node achieves a balanced combination of low-performance variations and higher average processing rates.

6. Conclusion

This paper proposes a novel in situ infrastructure for coupling Julia with HPC applications and presents a practical case study of applying two in situ Julia data analysis methods to atmosphere data for detecting extreme weather events and characterizing climate patterns. The data presented in this paper suggest five high-level takeaway messages: (1) the in situ infrastructure of coupling Julia with E3SM has insignificant overhead; (2) our developed SSW_GEV and PCA in situ modules in Julia are able to detect extreme weather events and characterize climate patterns with less than 26% of total module execution time; (3) a key performance factor of applying this in situ infrastructure is the total memory usage or pressure per node; (4) the development of this in situ infrastructure and consequent Julia data analysis modules can be highly independent; (5) future in situ Julia data analysis modules could be plugged-in without recompiling the whole HPC application. Having a comprehensive case study on this in situ infrastructure with production-level HPC application leads to better support for our future efforts on applying advanced data analysis methods to existing HPC applications.

Acknowledgement

Research presented in this paper was supported by the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project number 20200065DR and is released under LA-UR-22-31718. E3SM was obtained from the Energy Exascale Earth System Model project, sponsored by the U.S. Department of Energy, Office of Science, Office of Biological and Environmental Research. This research used resources provided by the Los Alamos National Laboratory Institutional Computing Program, which is supported by the U.S. Department of Energy National Nuclear Security Administration under Contract No. 89233218CNA000001.

7. References

[1] David Andrews, Conway Leovy, and James Holton. *Middle atmosphere dynamics*. Academic press, 1 1987. doi:https://doi.org/10.1038/294519a0.

[2] John Aycock. A brief history of just-in-time. *ACM Computing Surveys (CSUR)*, 35(2):97–113, 2003. doi:https://doi.org/10.1145/857076.857077.

[3] Mark P. Baldwin, Blanca Ayarzagüena, Thomas Birner, Neal Butchart, Amy H. Butler, Andrew J. Charlton-Perez, Daniela I. V. Domeisen, Chaim I. Garfinkel, Hella Garny, Edwin P. Gerber, Michaela I. Hegglin, Ulrike Langematz, and Nicholas M. Pedatella. Sudden stratospheric warmings. *Reviews of Geophysics*, 59(1):e2020RG000708, 2021. doi:https://doi.org/10.1029/2020RG000708.

[4] Andrew C. Bauer, Hasan Abbasi, James Ahrens, Hank Childs, Berk Geveci, Scott Klasky, Kenneth Moreland, Patrick O'Leary, Venkatram Vishwanath, Brad Whitlock, and E. W. Bethel. In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms. *Computer Graphics Forum*, 2016. doi:https://doi.org/10.1111/cgf.12930.

[5] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017. doi:10.1137/141000671.

[6] Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C Wilson, and Michael I Jordan. Streaming variational bayes.

Advances in neural information processing systems, 26, 2013. doi:https://dl.acm.org/doi/10.5555/2999792.2999805.

[7] Simon Byrne, Lucas C Wilcox, and Valentin Churavy. MPI.jl: Julia bindings for the message passing interface. In *Proceedings of the JuliaCon Conferences*, volume 1, page 68, 2021.

[8] Andrew J. Charlton and Lorenzo M. Polvani. A new look at stratospheric sudden warmings. part i: Climatology and modeling benchmarks. *Journal of Climate*, 20(3):449 – 469, 2007. doi:10.1175/JCLI3996.1.

[9] Dong Chen, David Irwin, and Prashant Shenoy. Smartsim: A device-accurate smart home simulator for energy analytics. In *2016 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 686–692. IEEE, 2016. doi:10.1109/SmartGridComm.2016.7778841.

[10] Hank Childs et al. A terminology for in situ visualization and analysis systems. *The International Journal of High Performance Computing Applications*, 34(6):676–691, 2020. doi:10.1177/1094342020935991.

[11] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM Journal on Scientific Computing*, 34(1):A206–A239, 2012. doi:https://doi.org/10.1137/080731992.

[12] Soumya Dutta, Natalie Klein, Li Tang, Jonathan David Wolfe, Luke Van Roekel, James Joseph Benedict, Ayan Biswas, Earl Lawrence, and Nathan Urban. *In Situ Climate Modeling for Analyzing Extreme Weather Events*, page 18–23. Association for Computing Machinery, New York, NY, USA, 2021. doi:https://doi.org/10.1145/3490138.3490142.

[13] Veronika Eyring, Sandrine Bony, Gerald A. Meehl, Catherine A. Senior, Bjorn Stevens, Ronald J. Stouffer, and Karl E. Taylor. Overview of the coupled model intercomparison project phase 6 (CMIP6) experimental design and organization. *Geoscientific Model Development*, 9(5):1937–1958, 2016. doi:10.5194/gmd-9-1937-2016.

[14] Nathan Fabian, Kenneth Moreland, David Thompson, Andrew C. Bauer, Pat Marion, Berk Gevecik, Michel Rasquin, and Kenneth E. Jansen. The ParaView coprocessing library: A scalable, general purpose in situ visualization library. In *2011 IEEE Symposium on Large Data Analysis and Visualization*, pages 89–96, 2011. doi:10.1109/LDAV.2011.6092322.

[15] Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 97–104. Springer, 2004. doi:https://doi.org/10.1007/978-3-540-30218-6_19.

[16] Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: a language for flexible probabilistic inference. In *International conference on artificial intelligence and statistics*, pages 1682–1690. PMLR, 2018. doi:https://doi.org/10.17863/CAM.42246.

[17] William F Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, et al. ADIOS 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561, 2020. doi:https://doi.org/10.1016/j.softx.2020.100561.

[18] Jean-Christophe Golaz, Peter M Caldwell, Luke P Van Roekel, Mark R Petersen, Qi Tang, Jonathan D Wolfe, Guta Abeshu, Valentine Anantharaj, Xylar S Asay-Davis, David C Bader, et al. The DOE E3SM coupled model version 1: Overview and evaluation at standard resolution. *Journal of Advances in Modeling Earth Systems*, 11(7):2089–2129, 2019. doi:https://doi.org/10.1029/2018MS001603.

[19] Jean-Christophe Golaz et al. The DOE E3SM coupled model version 1: Overview and evaluation at standard resolution. *Journal of Advances in Modeling Earth Systems*, 11(7):2089–2129, 2019. doi:https://doi.org/10.1029/2018MS001603.

[20] William Gropp and Ewing Lusk. User's guide for MPICH, a portable implementation of MPI, 1996.

[21] Amelia Henriksen and Rachel Ward. AdaOja: Adaptive learning rates for streaming PCA. *arXiv preprint arXiv:1905.12115*, 2019.

[22] Whitney K Huang, Michael L Stein, David J McInerney, Shanshan Sun, and Elisabeth J Moyer. Estimating changes in temperature extremes from millennial-scale climate simulations using generalized extreme value (gev) distributions. *Advances in Statistical Climatology, Meteorology and Oceanography*, 2(1):79–103, 2016. doi:https://doi.org/10.5194/ascmo-2-79-2016.

[23] Sascha Hunold and Sebastian Steiner. Benchmarking julia's communication performance: Is Julia HPC ready or full HPC? In *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 20–25. IEEE, 2020. doi:10.1109/PMBS51919.2020.00008.

[24] Sian Jin, Dingwen Tao, Houjun Tang, Sheng Di, Suren Byna, Zarija Lukic, and Franck Cappello. Accelerating parallel write via deeply integrating predictive lossy compression with HDF5. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2022. doi:10.1109/SC41404.2022.00066.

[25] Christopher Kelly, Sungsoo Ha, Kevin Huck, Hubertus Van Dam, Line Pouchard, Gyorgy Matyasfalvi, Li Tang, Nicholas D'Imperio, Wei Xu, Shinjae Yoo, et al. Chimbuko: A workflow-level scalable performance trace analysis tool. In *ISAV'20 In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pages 15–19, 2020. doi:https://doi.org/10.1145/3426462.3426465.

[26] Matthew Larsen, James Ahrens, Utkarsh Ayachit, Eric Brugger, Hank Childs, Berk Geveci, and Cyrus Harrison. 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'17, page 42–46, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3144769.3144778.

[27] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86. IEEE, 2004. doi:10.1109/CGO.2004.1281665.

[28] Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. 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*, CLADE '08, page 15–24, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1383529.1383533.

[29] Dhabaleswar K Panda, Karen Tomko, Karl Schulz, and Amitava Majumdar. The MVAPICH project: Evolution and sustainability of an open source production quality MPI library for HPC. In *Workshop on Sustainable Software for Science: Practice and Experiences, held in conjunction with Int'l Conference on Supercomputing (WSSPE)*, 2013.

[30] Yun Qian, Hui Wan, Ben Yang, Jean-Christophe Golaz, Bryce Harrop, Zhangshuan Hou, Vincent E Larson, L Ruby Leung, Guangxing Lin, Wuyin Lin, et al. Parametric sensitivity and uncertainty quantification in the version 1 of E3SM atmosphere model based on short perturbed parameter ensemble simulations. *Journal of Geophysical Research: Atmospheres*, 123(23):13–046, 2018. doi:https://doi.org/10.1029/2018JD028927.

[31] PJ Rasch, S Xie, P-L Ma, W Lin, H Wang, Q Tang, SM Burrows, P Caldwell, K Zhang, RC Easter, et al. An overview of the atmospheric component of the energy exascale earth system model. *Journal of Advances in Modeling Earth Systems*, 11(8):2377–2411, 2019. doi:https://doi.org/10.1029/2019MS001629.

[32] SCORPIO, a high-level parallel i/o library. `https://github.com/E3SM-Project/scorpio/`.

[33] SENSEI:Scalable in situ analysis and visualization. `https://sensei-insitu.org/`, 2021 (accessed April 6, 2022).

[34] Neil Thompson and Svenja Spanuth. The decline of computers as a general purpose technology: Why deep learning and the end of Moore's Law are fragmenting computing. *Available at SSRN 3287769*, 2018. doi:10.1145/3430936.

[35] David W Walker and Jack J Dongarra. MPI: a standard message passing interface. *Supercomputer*, 12:56–68, 1996. doi:10.1145/169627.169855.

[36] Yu Wang, Natalie Klein, Steven Morley, Vania Jordanova, Michael Henderson, Ayan Biswas, and Earl Lawrence. TributaryPCA: Distributed, streaming PCA for in situ dimension reduction with application to space weather simulations. In *2021 7th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-7)*, pages 33–39. IEEE, 2021. doi:10.1109/DRBSD754563.2021.00009.

[37] Zhuo Wang, Yujing Jiang, Hui Wan, Jun Yan, and Xuebin Zhang. Detection and attribution of changes in extreme temperatures at regional scale. *Journal of Climate*, 30(17):7035–7047, 2017. doi:https://doi.org/10.1175/JCLI-D-15-0835.1.

[38] Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. 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 '11, pages 101–109. Eurographics Association, 2011. doi:10.2312/EGPGV/EGPGV11/101-109.

[39] Jonathan Woodring, Mark Petersen, Andre Schmeißer, John Patchett, James Ahrens, and Hans Hagen. In situ Eddy analysis in a high-resolution ocean climate model. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):857–866, 2016. doi:10.1109/TVCG.2015.2467411.

[40] Shaocheng Xie, Wuyin Lin, Philip J Rasch, Po-Lun Ma, Richard Neale, Vincent E Larson, Yun Qian, Peter A Bogenschutz, Peter Caldwell, Philip Cameron-Smith, et al. Understanding cloud and convective characteristics in version 1 of the E3SM atmosphere model. *Journal of Advances in Modeling Earth Systems*, 10(10):2618–2644, 2018. doi:https://doi.org/10.1029/2018MS001562.

[41] Yuying Zhang, Shaocheng Xie, Wuyin Lin, Stephen A Klein, Mark Zelinka, Po-Lun Ma, Philip J Rasch, Yun Qian, Qi Tang, and Hsi-Yen Ma. Evaluation of clouds in version 1 of the E3SM atmosphere model with satellite simulators. *Journal of Advances in Modeling Earth Systems*, 11(5):1253–1268, 2019. doi:https://doi.org/10.1029/2018MS001562.