Embedded Domain-Specific Language and Runtime System for Progressive Spatiotemporal Data Analysis and Visualization

Cameron Christensen*
SCI Institute, University of Utah
Shusen Liu†
SCI Institute, University of Utah
Giorgio Scorzelli‡
SCI Institute, University of Utah
Ji-Woo Lee§
Lawrence Livermore National Laboratory
Peer-Timo Bremer¶
Lawrence Livermore National Laboratory
Valerio Pascucci‖
SCI Institute, University of Utah

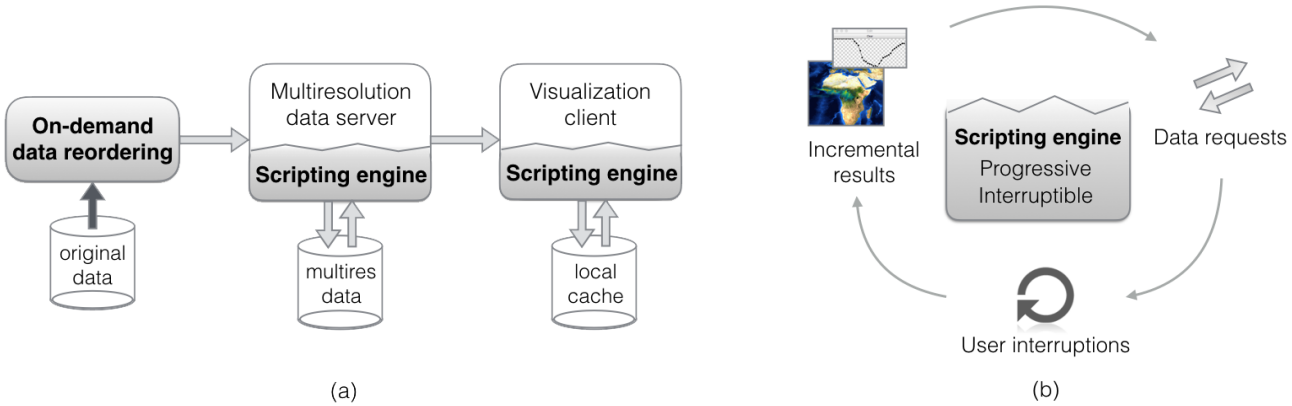


Figure 1: Our interactive analysis and visualization framework exploits progressive computation and seamless local or remote execution of embedded domain-specific language (EDSL) scripts to provide a highly flexible platform for the exploration of large-scale, disparately located data. As illustrated in the system pipeline (a), what differentiates the proposed system from existing techniques is the ability to utilize an embedded domain-specific language to specify data analysis workflows. The execution model of the runtime is shown in (b). The interactive runtime continuously processes data requests, publishes incremental results, and responds immediately to user input.

Abstract

As our ability to generate large and complex datasets grows, accessing and processing these massive data collections is increasingly the primary bottleneck in scientific analysis. Challenges include retrieving, converting, resampling, and combining remote and often disparately located data ensembles with only limited support from existing tools. In particular, existing solutions rely predominantly on extensive data transfers or large-scale remote computing resources, both of which are inherently offline processes with long delays and substantial repercussions for any mistakes. Such workflows severely limit the flexible exploration and rapid evaluation of new hypotheses that are crucial to the scientific process and thereby impede scientific discovery. Here we present an embedded domain-specific language (EDSL) specifically designed for the interactive exploration of large-scale, remote data. Our EDSL allows users to express a wide range of data analysis operations in a simple and abstract manner. The underlying runtime system transparently resolves issues such as remote data access and resampling while at the same time maintaining interactivity through progressive and interruptible computation. This system enables, for the first time, interactive remote exploration of massive datasets such as the 7km NASA GEOS-5 Nature Run simulation, which previously have been analyzed only offline or at reduced resolution.

Keywords: Streaming, Analysis, Big Data, Climate, Dynamic, Remote.

Index Terms: H.3 [INFORMATION STORAGE AND RETRIEVAL]: Information Search and Retrieval—Online Information Services; J.2 [PHYSICAL SCIENCES AND ENGINEERING]: Earth and atmospheric sciences—Mathematics and statistics

1 Introduction

Interactivity has long been a desirable trait for most scientific visualization systems. Instantaneous feedback in response to user input enables flexible data exploration and analysis and streamlines the hypothesis-to-evaluation loop, which is vital for data-driven scientific discovery. Most previous work has focused either on fast queries (i.e., FastBit [41]), or interactive rendering (i.e., iso-surface extraction [6]), rather than the whole process that transforms the raw data into visualized images. Yet as our ability to generate large datasets and our reliance on distributed storage grows, both data acquisition and processing time can severely limit scientists' ability to conduct effective data-driven experiments. Describing even comparatively simple analysis tasks such as averages or comparisons can quickly become nontrivial when these tasks involve multiple data sources, remote computation, or data of different formats or resolutions. The resulting scripts and solutions are typically customized for the specific analysis, and often rely on manual steps such as file transfers. Furthermore, due to the overwhelming size of the data, users likely need to run analyses on a high-performance computing machine and wait for the results. The inherent processing latencies can be prohibitive, and the workflows can be difficult to adapt. Mistakes

*e-mail: cam@sci.utah.edu
†e-mail: shusenl@sci.utah.edu
‡e-mail: scrgiorgio@gmail.com
§e-mail: lee1043@llnl.gov
¶e-mail: bremer5@llnl.gov
‖e-mail: pascucci@sci.utah.edu

at any point in these workflows can carry a heavy penalty, often requiring repetition of significant parts of these time-consuming processes. This toil severely limits a scientist's ability to conduct effective data-driven experiments.

In the proposed work, we aim to address the challenges of analysis and visualization of massive, diaparately located datasets by utilizing progressive algorithms in recognition of the utility of incremental computation results for the realization of a genuinely interactive data analysis and visualization environment. The key to streamlining the data access and aggregation lies in the ability to allow the user to focus on high-level logic while automating low-level data operations. To this end, we introduce an embedded domain-specific language (EDSL) to hide such low-level complexity from the user. For data analysis workflows created using the proposed EDSL, the user can focus on operations directly associated with the analysis, such as statistical operations and comparison, whereas details such as the source location, data transfer, file formats, and grid resolutions are automatically handled by the language runtime system. To speed up data processing, the system accesses and transfers the least amount of data possible for the given computation. The generality of the EDSL allows great flexibility in its interpretation, enabling a suitable runtime system to exploit task parallelism appropriate for large, dispersed data. The design of the runtime system focuses on multiresolution storage and visualization such that preliminary results can be obtained without significant delay, followed by progressive refinement. Our key contributions are:

1. An embedded DSL based on JavaScript that provides a simple and abstract description of sophisticated analysis and visualization workflows;

2. The corresponding runtime system that executes a given workflow in an interruptible, progressive manner and enables dynamic selection of various computational parameters; and

3. An end-to-end pipeline for automatic conversion and caching that enables transparent multiresolution access to distributed datasets of different formats.

2 Related Work

In this section, we examine related work and discuss how it compares with our efforts.

General Integrated Visualization Environment. To lower the access barriers for complex visualization techniques, integrated visualization systems, such as VisIt [7] and Paraview [2], have been introduced to allow domain scientists to easily visualize their datasets using different algorithms, such as iso-surfaces, volume rendering, and streamlines. However, even though these integrated systems provide extensive visualization capabilities and customized scripting, it is necessary to manually specify data types, and explicitly define the exact data structures that will be produced by the built-in scripts. Even simply combining data of different resolutions is nontrivial. Furthermore, these applications are not capable of displaying incremental updates necessary to maintain interactivity, and therefore entail workflows that involve scripts and processes with many of the same characteristics as the offline workflow. Essentially, the exploratory analysis process suffers from high "latencies" in the sense that parameter modifications or other changes require potentially lengthy reevaluations.

Domain-Specific Visualization Systems. Besides the general integrated visualization environment, many systems focus on a specific domain such as climate analysis (UV-CDAT [35], DV3D [27]). By concentrating on a more specific application, these systems usually have fewer but more specialized capabilities. For example, UV-CDAT is designed for climate data visualization. By incorporating many standard analysis and visualization techniques for climate data, the scientists have an easy-to-use tool that is adequate for most visualization needs. However, for a modified workflow, as pointed out by our collaborator, the scientists are often required to write customized code to fill in missing features in these domain-specific visualization systems. He suggests the proposed embedded domain-specific language can tremendously simplify and streamline such a process.

Remote Data Access. Scientific analysis tools such as VisIt and Paraview enable complex workflows but struggle with remote data, and setting up the workflows can be difficult. Local data analysis tools can benefit from protocols such as OPeNDAP [1] that provide local access to remote data, but these protocols are tied to the same limitations as the underlying fixed-resolution data formats they serve, and do not do anything to facilitate the hierarchical access needed to scale interactive systems to extremely large data sizes.

Workflow Management Systems. There exist sophisticated distributed workflow management systems like Pegasus [11] and Kepler [25], but these are defined largely for offline use for which robustness to failures, data provenance, workflow abstraction, and reliability are the key concerns, and their use is not amenable to the requirements of an interactive system.

Domain-Specific Languages. Languages such as Diderot [18] and ViSlang [33] are specialized DSLs designed for visualization and do not handle remote data. Our work is intended for data processing of possibly remote data often used for the analysis and comparison of scientific datasets, rather than focused purely on visualization-specific tasks. Other DSLs, such as Ebb [3] and Simit [19], are designed for physical simulation while abstracting execution environments to enable CPU, GPU, and parallel execution of common code. Others, such as Vivaldi [8], combine a specialized DSL for visualization with a mixed execution model. Our DSL and associated runtime enable interactive exploration through progressive remote data access and interruptible analyses rather than reducing total computation time by utilizing such hybrid execution backends. The results of our processing nodes could be used as input for visualization-specific DSLs such as Vivaldi or Diderot, enabling these languages to be used for the visualization of a wider range of local and remote data. Languages such as Ebb or Simit could be useful to perform more efficient server-side computation for which interruptibility may be less desirable than fast computation.

Runtime Loop Optimizations. Portability and optimization of analysis programs is an issue that has been addressed with the use of directives such as provided by OpenACC [5] and OpenMP [10], cross compilers that create optimized versions of some other code [13, 26], and wrappers to provide a specific specialized set of portable optimized functions. Thrust [17], RAJA [32]. Kokkos [14] provides vector libraries to manage multidimensional arrays with polymorphic layouts and map those operations to fast manycore implementations. Overall, these works focus on providing specific optimizations of existing code rather than enabling a simple semantic for scientists to express iterative computations.

3 Background

In this work, we strive to present the user with intermediate or partial results quickly and then progressively refine them. One aspect we exploit is the spatial resolution of data. As a result, we build the proposed system on top of an existing multiresolution data storage and visualization framework. The ViSUS Visualization Framework [30, 31] enables streaming access to arbitrarily high-resolution imagery through the use of an efficient multiresolution data reordering based on the hierarchical Morton Z-order space-filling curve [29]. As illustrated in Fig. 2, by utilizing a multiresolution data layout, data can be loaded and visualized at coarse resolution, then successively refined as more data is streamed into the system. Without loading the full dataset, preliminary results can be rapidly obtained.

Localized queries are also optimized using the Morton data order, the layout of which naturally favors queries of rectilinear subregions.

Multiresolution data formats range from simple octrees to more complex or distributed schemes such as [16, 37]. The proposed DSL and runtime of this work are logically separated from the underlying multiresolution data format used by the system. The data format could be replaced by one of these other multiresolution approaches and the work would still retain most of the benefits provided by the proposed DSL. Details of the multiresolution and data reordering algorithms are outside the scope of this work, and readers are encouraged to explore the references above for more information. Similar to other integrated visualization systems (e.g., VisIt or Paraview), the ViSUS framework also includes a set of common visualization algorithms, such as volume rendering and iso-surface extraction. The framework is multithreaded and implements a directed acyclic graph, message-based dataflow pipeline such that messages can be "published" by a given node to connected nodes. The multithreaded implementation enables visualization and computation tasks to be carried out simultaneously.

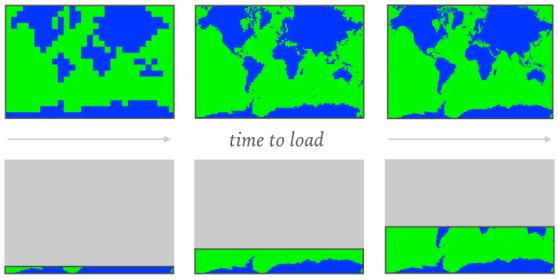


Figure 2: Illustration of multiresolution data loading compared to loading from a "flat" row-major format. Using multiresolution IDX, coarse resolution data can be loaded in much less time, providing quick preliminary results.

4 Method

In this section, we discuss the design and implementation of the embedded domain-specific language (EDSL) and complementary runtime system. The overall system is illustrated in Fig. 1(a). The pipeline works as follows. An EDSL script is executed incrementally on the visualization client. When data is needed by the script, the client requests it from the multiresolution server, which first checks its local cache and if found immediately fulfills the request. If cached data is not found, the server requests the on-demand data reordering service to produce a multiresolution version of the data, which is cached and sent to the client. The visualization client produces results incrementally as they are computed. What differentiates the proposed system from existing progressive visualization techniques is the ability to utilize an embedded domain-specific language to specify data analysis workflows that hide the complexity originating from combining multiple input sources and spatial resolutions, and an interruptible script processing engine that facilitates progressive computation. Such a design provides the user tremendous expressive power to write custom, reusable analysis workflows suitable for rapid data exploration.

The embedded language introduced next is designed to permit the types of interpretation necessary for an interactive system without compromising expressiveness or accuracy, and the runtime system and scripting engine introduced in Section 4.2 enable interactive execution of these scripts.

4.1 Data Processing Embedded DSL

Our goal is to provide a simple and abstract language for describing rich data processing tasks that relieves users from having to deal with mundane tasks such as data import and resampling (also called "regridding") and allows for incremental execution suitable for an interactive environment. We assert that necessary modifications to the host language can be limited to three aspects, discussed in the following sections, which are sufficient to facilitate interactive evaluation of generic data processing scripts.

1. A new built-in data type that abstracts the common modalities of scientific data (e.g., scalar or vector field data) and can be used directly as a first class citizen of the language without regard to format, resolution, or location of the underlying data;

2. A hinting mechanism to facilitate incremental production of the results of ongoing computations (i.e., long-running scripts) by indicating to the runtime system appropriate opportunities at which the current state of the computation can be shown; and

3. A generic multidimensional iterator for loops that can be performed in any order (e.g., for computing an average) that permits nonlinear evaluation of the loop body by the runtime system such that incremental results potentially converge faster toward the final result, and allows for parallelization of these loops.

In the remainder of this section, we will explain each language addition in detail and present a simple example script to illustrate them.

Abstract Data Type. An abstract data type is necessary in order to enable spatiotemporal data manipulation using a uniform and generic interface without regard to format, resolution, or location. The use of this type avoids embedding details in the data processing scripts concerning the management of the underlying data. The runtime system will handle data loading, resampling, and conversion to a common format. We chose to make this a built-in type of the EDSL to enable features such as operator overloading that otherwise might not be feasible in the host language.

The specific methods provided by our EDSL include statistical summary operations such as *mean* and *variance*, multifield operations that perform element-wise amalgamation such as *average* and *maximum*, and operations such as *convolve* that involve some degree of global dataset-wide access. A complete listing of the abstract data type methods is provided in the addendum. Operator overloading is provided to enable natural expression of element-wise operations between fields of the new data type or with scalars. These methods are sufficient for constructing arbitrarily sophisticated scripts for the computation of temporal averages, rank correlations, image segmentations, maximum intensity projections, and other types of output used in scientific data analyses.

Explicit Data Publishing Hints. Streaming algorithms provide incremental results based on incoming data that represent the best possible computation for the currently available input. These snapshots of ongoing computations present the user with an approximation of the final results of long-running operations, enabling errors to be caught and addressed much sooner. Feedback is particularly desirable for users of an interactive system, but for script-driven analysis the best times to show these incremental results are not always apparent. Attempting automatic determination could result in showing incorrect or undesirable results, such as when a script utilizes an output variable as a temporary. In order to show progressive results for streaming computations while avoiding output at the wrong time, we introduce the *doPublish* primitive operation, which indicates appropriate times for the scripting engine to send the current computation results, designated *output* in Listing 1 below, to the visualization system. Using this primitive enables the corresponding workflows to be progressive with partial results being computed and updated continuously. The *doPublish* primitive has no

effect on the computation itself, and can be safely ignored, enabling the runtime system to refresh output presented to the user at intervals suitable to maintain interactivity.

Generalized Multidimensional Iterators. To complement the progressive asynchronous updates enabled by *doPublish*, we introduce an iterator for order-independent loops called *unordered*. This generalized facility allows for a variety of beneficial execution methods to be utilized by the runtime system, and provides for the expression of multidimensional loops that is both elegant and flexible. The *unordered* primitive accepts as parameters the name of the variable to be used as an index inside the loop and the extents of the loop iterator. Loop indices are considered constant within the body of the computation. The result of the loop should be the same regardless of the order of execution (except for floating point differences that would be expected to occur anyway), and it is considered a bug for the user to construct an *unordered* loop body that depends on some particular order of execution. In addition to parallelization, other useful interpretations of *unordered* loops are described in detail in Section 4.2.

The proposed EDSL described in this section primarily consists of JavaScript extended with these carefully chosen primitives and a new built-in data type for scientific data. This new EDSL allows users to express common workflows in an abstract manner, suitable for interactive execution. In Section 4.2, we introduce a runtime scripting engine designed for progressive, interactive execution of these EDSL scripts for computations over arbitrarily large, disparately located datasets. An example script is presented next that illustrates the EDSL features described above.

Listing 1: EDSL script for incremental computation of a temporal average using hourly data from the 7km GEOS-5 Nature Run simulation. Notice the ability to succinctly express a significant operation without explicitly addressing input format, resolution, dimension, or output type.

```
//Computes running average
field = 'TOTSCATAU';                  //aerosol scattering
start = query_time;                   //current time
width = 720;                          //720 hours (30 days)

output=Array.New();                   //initialize output
var i=0;
unordered(t,[start,start+width])      //1d iterator, index t
{
  f=input[field+"?time="+t];          //read field at time t

  //critical section for running average:
  //average and count must be updated atomically
  {{
    output += (f-output)/(i+1);       //Welford's method
    i++;
  }}

  doPublish();                        //show current result
}
```

Example Script. Listing 1 shows an example of a basic incremental computation using the proposed EDSL. The script makes use of Welford's method [20, 40] to compute a monthly average of hourly temporal climate data. The script is able to express in very terse terms a significant operation without the user needing to explicitly address input formats, data resolution, or output type. Notice the use of the overloaded arithmetic operators +, -, += in the statement *output += (f-output) / (i+1)*. For this expression, *output* and *f* are members of our new abstract data type representing the current state of the incremental average computation and the field at the current timestep of the iteration, respectively. The *unordered* loop could be interpreted just like a normal *for* loop, but using this facility enables other execution methods as described in Section 4.2. The double opening and closing brackets around the two statements designate a critical section. If the loop were executed in parallel by the runtime system, this delineation would be necessary to ensure correctness by atomically updating the current output and running count. The call to *doPublish* allows for incremental display of the result.

For comparison, a similar computation in the VisIt expressions Python-based EDSL would require creating a specific class template structure in which the user must explicitly define output type and dimensions and manually create the VTK arrays to be computed by the script. The VisIt EDSL contains a specific function for computing temporal averages, *average_over_time*, but this type of specialization, in addition to being unnecessary in the proposed EDSL, does not facilitate progressive display of in-progress results that are a focus of the proposed system in order to provide quick and preliminary visualization.

Please refer to the Appendix of this work for a comprehensive specification of the EDSL.

4.2 Progressive Runtime System

Now we present the complementary runtime system for the EDSL presented in section 4.1, which incorporates an interruptible script processing engine to evaluate scripts in an interactive manner by enabling tuning of any necessary parameters in order to enable computations to be performed quickly and incrementally. Through the genericity of the EDSL, the runtime system also enables direct and transparent transition from a local execution to a distributed workflow, including server-side execution and caching. To demonstrate the scripting engine presented here, we wrote our own JavaScript interpreter, used by the engine to directly execute scripts without any compilation to byte code or significant optimization. Type-checking is enabled at runtime using exceptions, which display the problematic line of the script and a detailed error message to the user to enable debugging. The presented runtime system utilizes techniques such as multiresolution streaming and low-discrepancy sampling to produce progressive results from streaming input data. The goal of the system is to minimize the tradeoffs between accuracy and speed while continuously providing useful results during interactive data exploration.

The following paragraphs describe the features of the runtime system that enable practical data exploration through interactive interpretation of EDSL scripts, including implementation of the built-in scientific data type, design of the progressive scripting engine, making effective use of order-independent loops and parallelization, and our method for remote or distributed script processing.

Multiresolution Streaming. Our runtime system reads and caches input data using a lossless multiresolution format that provides efficient coarse-to-fine data loading and much faster access to local regions of interest compared to traditional row- or column-major order data [29]. In order to provide transparent access to multiresolution data from other data formats, an on-demand reordering facility is presented in section 4.3. Multiresolution data can be used to provide fast cursory computations by displaying the result of an initial coarse-resolution execution while refining it to provide more details when they are needed. The results of computation using coarse-resolution data can also be surprisingly accurate. One of our case studies in section 5 compares these results by performing the same computation at different resolutions. See Fig. 11 in that section for more details.

Built-in Data Model. The runtime scripting engine utilizes a fast C++ *Array* type to provide efficient implementations of the operations defined for the EDSL built-in data type, similar to the *numpy* package for Python [39]. Since datasets can be manipulated without regard to their location, the runtime system uses additional metadata associated with a script to map its inputs to their corresponding local or remote data locations. Data read from remote locations is automatically cached on the local system, and the results of a given script

execution can be cached as well, allowing comparison of new results with previously computed data. Finally, the EDSL specifies element-wise operations that can be performed independent of the resolution of the operands. Variables of this type must be implicitly resampled to the same resolution to be combined or compared. By default, the scripting engine uses upsampling to the largest resolutions in each dimension of the given operands and linear interpolation for resampling. These methods, however, can be changed by the user at any time without modification to the original script. Resampling data in order to perform computations among different models is a serious impediment for scientists, and we present a powerful case study in section 5 that demonstrates our system's ability to effortlessly facilitate comparison of multiple climate ensembles.

Progressivity and Incremental Results. When the runtime script interpreter encounters *doPublish* in a script, it can produce, or "publish", the current state of an ongoing computation to provide the user with important and timely feedback. Such a call can be safely ignored by the downstream visualization without adverse effects to the computation, enabling results to be displayed at suitable intervals to maintain interactivity. The scripting engine implements *doPublish* as an asynchronous callback that creates a copy of the current computation output to be displayed by the visualization client. If a previously published result has not yet been displayed by the visualization system, that result is simply replaced with the new output, ensuring smooth performance of the rest of the system while allowing script execution to continue uninterrupted.

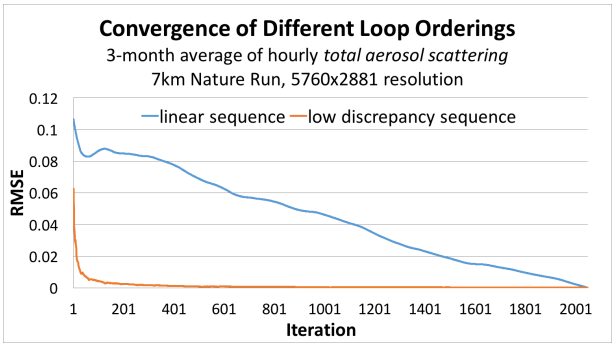
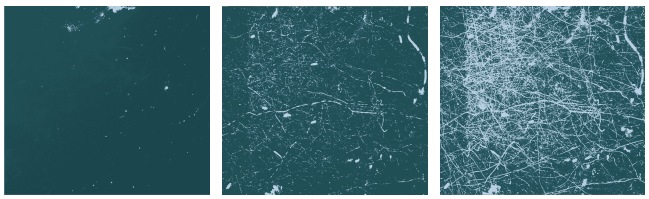


Figure 3: Results of a temporal average computation (Listing 1) via two orderings for the inner loop. The error (plotted as RMSE) between the *precomputed result* and the incremental result decreases quickly when utilizing the low-discrepancy van der Corput sequence of timesteps versus a simple linear sequence.

Loop Order and Parallelization. The EDSL presented in section 4.1 introduced the *unordered* primitive to allow explicit declaration of order-independent multidimensional loops. These calculations are common in scientific data analyses, yet their properties are rarely exploited. For many iterative calculations, using an input ordering with low discrepancy can lead to faster convergence of successive iterations to the final solution compared to a simple linear sequence. The desirable qualities of a low-discrepancy ordering are *uniformity* and *incrementality*, such that samples are evenly distributed over the given range, and decent coverage will have been achieved if the processing is terminated at any point in the sequence [24].

Consider the incremental average script from Listing 1. This script could simply use a *for* loop, but since the final result of the computation does not depend on the order of loop iterations, we can choose a superior ordering that converges significantly faster. Fig. 3 illustrates the difference of using a linear ordering compared to the low-discrepancy sequence introduced by van der Corput [12] for the incremental result of this computation. For higher dimensional iterators, the Halton sequence [15] can be used. Fundamentally, any evaluation order can be chosen at runtime for these loops, allowing the flexibility to choose different orderings, for example to maximize the use of cached data.



(a) (b) (c)

Figure 4: Result of 100 iterations (of 1000 total) for calculating maximum intensity projection of microscopy volume. Each iteration adds a 2d slice. (a) Linear order. (b) Low-discrepancy order. (c) Final MIP.

Another example of the utilization of low-discrepancy loop orderings is shown in Fig. 4. High-resolution microscopy is used by neuroscientists to examine cortical tissue samples to study the connectivity of the brain. To aid in clarifying the imaged neurons, which may not always be obvious from the 3d visualization, a *maximum intensity projection* is often used. This type of projection accumulates the maximum intensity value of each voxel along a given axis and presents a 2d image of the result. The incremental computation of this projection is demonstrated using both linear and low-discrepancy orderings. The results after processing the first 100 slices using each ordering are shown in Fig. 4 along with the final result of the computation. Note in this case how much faster the results of the incremental computation approach the final result when using the low-discrepancy sequence. Using this technique interactively enables faster and more dynamic comprehension of custom regions of interest within massive microscopy volumes.

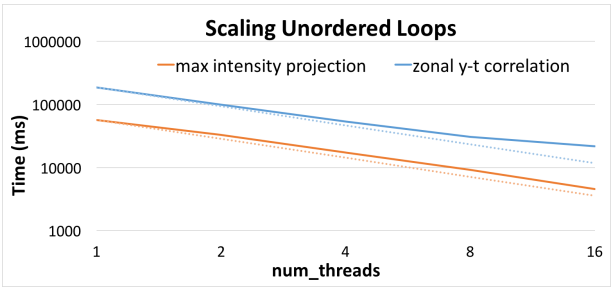


Figure 5: Comparison of parallel unordered loop execution for increasing thread counts for two algorithms: maximum intensity projection and zonal rank correlation. Dashed lines indicate perfect scaling. Tests conducted on a 16-core Intel Xeon E7-8890 v3 @ 2.50GHz running openSUSE 13.1 using locally cached data.

Finally, parallelization of unordered loops potentially enables faster evaluation by executing multiple iterations simultaneously. Parallel execution may require more delicacy in the implementation of an analysis script, and the double bracket {{ ... }} notation was incorporated into the EDSL in order to denote a critical section. Code within these sections will be executed atomically with respect to the other threads, ensuring correct functionality of parallel code without overly complicating or cluttering the resultant scripts. The number of threads is controlled by the runtime system and provided to the EDSL script, so scripts can be scaled to the available system resources without modification.

Parallel loop execution is implemented in the scripting engine using a thread pool for each unordered loop and assigning the work of one iteration to each thread, with a shared context of global variables and a thread-local context for variables introduced in the iteration block. Critical sections are facilitated by using a shared

lock per loop. This strategy enables nested *unordered* loops, but one should beware of the potential explosion of tasks and consider rewriting the loop to instead utilize a multidimensional version of the unordered iterator. As shown in Fig. 5, parallelizing the execution of order-independent loops can provide a modest speedup even for relatively naive algorithms. Permitting simple scripts to make better use of processing resources is a benefit to the user that permits more practically useful interactive data exploration.

Server-Side Processing. The multiresolution data server contains an identical version of the scripting engine used by the visualization client (see Fig. 1). Server-side processing can be utilized to perform computations using remote resources and thereby reduce data transmission. For example, when combining many ensemble members into a single average, the amount of data to be sent to the client can be dramatically reduced by first combining the inputs on the server and then sending only the result to the client. On the other hand, if server-side resources are scarce or in high demand, it may be more efficient to transmit data directly to the client, perhaps at lower resolution to reduce network bandwidth. The runtime system specifies whether or not to perform a computation remotely without requiring any modification to the input script, enabling a single script to be executed on either the client or the server. Multiple scripts can be incorporated within larger dataflows to mix both client- and server-side processing. The location on which to execute a computation is currently specified by the user on a per-script basis, but future work will aim to address automatic selection based on available resources.

For the implementation of the runtime system, we extended the ViSUS framework mentioned in section 3 to include a new scripting engine that enables execution of generic EDSL scripts in a manner that is both progressive and accurate by making effective use of multiresolution data, asynchronous output, flexible iterator orderings, remote computation resources, and parallelization. A novel data ingest system was also added to automatically resample the various input datasets specified in the script to a common domain during I/O. High-level support was added to the UI for the selection of the various runtime parameters, such as the default order used for multidimensional iterators.

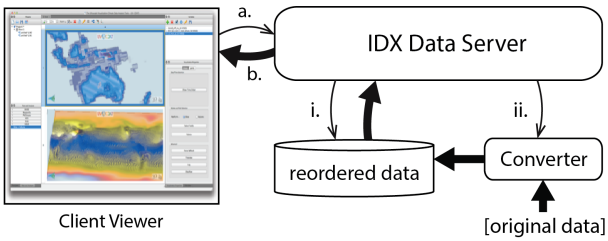


Figure 6: Data server with on-demand conversion. Data movement is shown with thick arrows, requests with thin arrows. When data is requested (a), the data server first checks the cache (i), and if not cached the requested data is converted on-the-fly (ii) and sent to the client (b).

4.3 On-Demand Data Reordering

Although some simulation frameworks have adopted multiresolution formats as their default output [21], many existing datasets are not stored in this fashion and must be converted prior to use.

On-Demand Conversion. We propose a data reordering service that converts requested data on-the-fly to the multiresolution format utilized by our data analysis runtime system. This system operates transparently to the client, enabling access to data from other formats without requiring explicit preprocessing.

Fig. 6 shows an overview of the system. When a data request is made by the client application to the multiresolution IDX data server (a), the server first checks its cache for the data (i), and if found, the request is fulfilled directly (b). Otherwise, the server makes a call to the on-demand service (ii), which reads the full-resolution data and writes the multiresolution version to the data cache. This lossless reordering of the original data is now sent to the requester (b) and is also available for future requests by other users. The cache size is maintained by periodically removing least recently used data when the size grows beyond a specified maximum level. Data reordering is a computationally light task, and the time required to convert a given volume is dominated by the time to read the original data and to write the reordered version. In general, the on-demand conversion system does not introduce a significant overhead, since the data would have to be downloaded anyway, and the initial conversion time may be amortized over many future requests. Reordered data facilitates interactive analysis and visualization that would in many cases be impossible if the data remained in its original format. The transparent, on-demand data reordering service described in this section is utilized for climate analysis in the examples described in Section 5. This implementation is briefly described next.

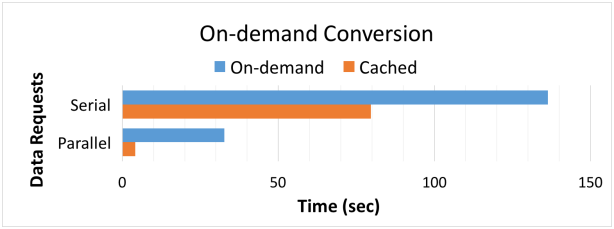


Figure 7: Computation time when input data is converted on demand versus already cached on the server. Temporal average of daily data (90 timesteps) from NIMR HadGEM2-AO "Historic". Local caching disabled. Each timestep is 32-bit floating point, resolution 192x143x8. Our progressive environment revealed serious and previously unnoticed errors in the original data.

On-demand Reordering for Climate Data. We have integrated the proposed on-demand service at part of the Earth System Grid Federation (ESGF) at Lawrence Livermore National Laboratory (LLNL). The service provides for converting both local and remote climate datasets to the multiresolution IDX format. The reordering service is implemented as a python-based web service with read-only access to hundreds of terabytes of (possibly remote) climate data stored in the NetCDF format [34]. The typical method by which a user of climate data federated by ESGF acquires new data is to first search for the desired dataset using the ESGF search page, then to manually select and download the datasets to be studied. These data may be very large and contain many fields not needed for the desired experiment, wasting time and local storage space. The multiresolution datasets provided by our service incorporate all fields and the entire time span of a given dataset, but no actual data is converted until it is specifically requested. This efficiency makes it simple for scientists to add or remove an unexpected field from their computations without converting unnecesary data. Fig. 7 shows the time required to compute a seasonal temporal average (see Listing 1) when data is converted on-demand versus when it already exists in the server cache. Note that client-side caching was disabled for this test.

5 Results

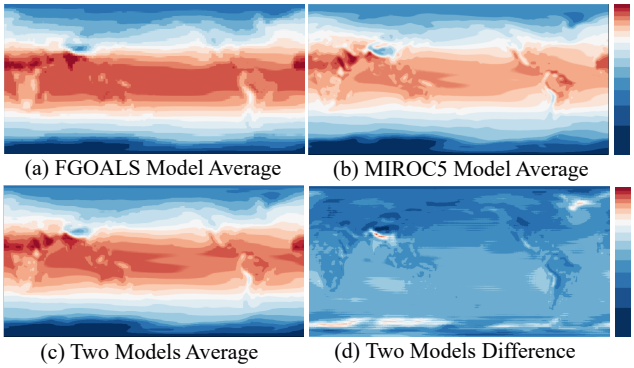
In this section, we demonstrate the usability of the proposed system for various analysis scenarios in real-world scientific applications. All scripts for constructing these workflows can be found in the Appendix of the paper.

5.1 Climate Simulation

Global climate research has become a major undertaking of many governments and organizations in order to understand the primary

causes of the unusual warming observed over the past several decades, as well as to determine the extent to which this warming can be mitigated by changes in human behavior such as a reduction in carbon dioxide emissions.

According to domain scientists, as computational capability increases, these models become more sophisticated and the size of climate simulation output grows dramatically (up to Petabytes for simulations with extremely high spatial and/or temporal resolution). The increasing size and complexity of climate datasets have placed a huge burden on scientists to effectively perform analysis and visualization tasks. By utilizing the proposed system, scientists can streamline and automate large amounts of manual operations such as downloading, converting, or resampling. The analysis tasks themselves can be concisely expressed in the reusable and easy-to-understand EDSL. For each type of analysis task, the same script can be used directly with only minor changes (or none at all). The significant time-savings and convenience provided by our framework enables scientists to focus on core analysis tasks, and encourages them to experiment more. This experimentation allowed an error in a widely used public dataset to be discovered (see Fig. 9 from the Annual Zonal Average example below).



(a) FGOALS Model Average (b) MIROC5 Model Average

(c) Two Models Average (d) Two Models Difference

Figure 8: The comparison between climate simulation model ensembles.

Multimodel Ensemble Comparison. There are numerous climate models from different countries and institutions. One of the important tasks of climate research is to validate these models against historical observations as well as compare them with each other [36]. These models can then be used in experiments that try to predict future climate under a variety of conditions such as increased or decreased anthropogenic emissions. For each model (and a given experiment), a collection of runs is generated, each with different parameters and/or initial conditions. Such a collection is often referred to as an ensemble. These models are created by different institutions with different computational resources, and therefore the grid resolution of the output data is usually different. As a result, resampling is necessary for comparison. Compared to the tedious manual workflow that is often adopted by domain scientists, our system utilizes remote data directly, streaming even very large data interactively at reduced resolutions, refining as necessary, and implicitly resampling the requested datasets to a common resolution for proper comparison.

In Fig. 8(a), we visualize the temperature average from an ensemble of the FGOALS model (12-run ensemble). The average of the same experiment for the MIROC5 model (12-run ensemble) is shown in Fig. 8(b). The average and difference of these two models are illustrated in Fig. 8(c) and (d). As we can easily see from Fig. 8(d), these two models demonstrate the greatest divergence in the area between the Tibetan plain and the Indian subcontinent. By using our system, such observations can be obtained on-the-fly without tedious data conversion and grid resampling.

Annual Zonal Average. Another interesting analysis called a zonal average can be applied to climate data. The temperature field zonal average in Fig. 9(a) shows the daily data for a whole year summarized in one figure. The average for the entire line of longitude is computed at each latitude for daily data. In the plot, along the x-axis each vertical line corresponds to one day's planetary average. As we can see in Fig. 9(a), the temperature corresponding to each latitude changes over time, indicating seasonal variation.

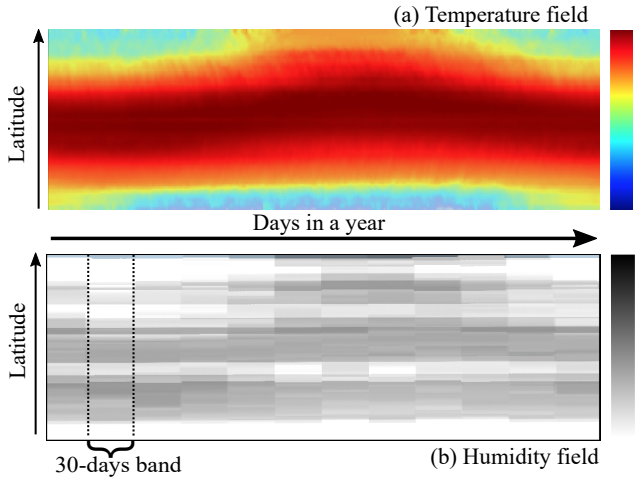


Figure 9: Annual zonal average of temperature and humidity. In (a), the daily spatial temperature average changes as we move along the temporal axis, which illustrates the change of seasons in a year. In (b), the duplication error in the humidity data is indicated by the bands along the temporal axis.

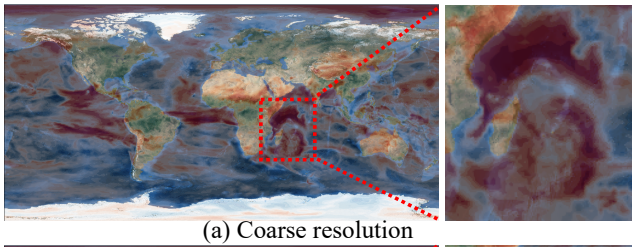
By utilizing our system for cursory exploratory analysis, the scientist also revealed a serious set of errors in the daily 3d data from NIMR (the Korean National Institute of Meteorology Research). As illustrated in Fig. 9(b), the zonal average shows unnatural bands in the horizontal (temporal) direction, which indicated unchanging daily data for each 30-day period. In this particular ensemble, it turns out that for each month, a single day's data was erroneously duplicated for the entire month. Once we observed the flaw in one field, it was trivial to check the other 3d fields that also exhibited the error by simply changing the variable name in the script. The on-demand data conversion system transparently converted these additional fields, which would have otherwise required manual download to be examined. In addition, the low-discrepancy ordering of the unordered loops used to generate the zonal averages ensured that incoming data provided the best possible incremental representation of the entire zone. What might have required significant manual effort and hours of computation was able to be achieved in minutes and at a glance.

Rank Correlation Analysis. Next we demonstrate the efficacy of our EDSL and runtime for performing a more complicated type of analysis. Studying correlation plays an important role in analyzing climate data. Rank correlation can be used for measuring relevance between different variables, examining a model's performance by comparing its variables to corresponding observations, and even comparing different regions in a single model. According to our collaborator, rank correlation [4,23,28] is a widely used but relatively new technique for climate analysis that is rarely implemented in domain-specific analysis tools. Instead, scientists must manually write code to compute these correlations. Listing 2 shows the main loop of an EDSL script that incrementally computes rank correlation. Notice the use of overloaded operators for summation (+=) and scaling (/) of the 3d fields. The algorithm is based on Welford's method to compute the running variances of the two fields that are necessary to calculate the rank correlation.

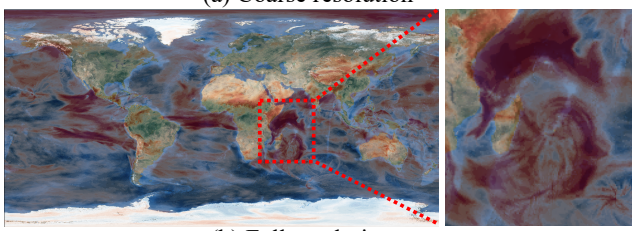
Listing 2: EDSL script for incremental computation of Pearson's rank correlation using hourly 3d data from the 7km GEOS-5 Nature Run simulation.

```
//Pearson's rank correlation of two variables over time
var i = 0;
unordered(t,[start,start+width])
{
    f = dataset1[field1+"?time="+t[0]];
    g = dataset2[field2+"?time="+t[0]];

    //critical section:
    // update running average, variance, and correlation
    // w.r.t. their current values and the given index
    {{
        var oldMf = Mf;
        var oldMg = Mg;

        //running average
        Mf += (f-Mf)/(i+1);
        Mg += (g-Mg)/(i+1);

        //running variance
        Vf += (f-Mf)*(f-oldMf);
        Vg += (g-Mg)*(g-oldMg);

        //running correlation
        Vfg += ((oldMf-f)*(oldMg-g))*((i+0.0)/(i+1.0));
        var Sf = Array.sqrt(Vf/i);
        var Sg = Array.sqrt(Vg/i);
        output  = Sfg/(Sf*Sg*i);
        i++;
    }}

    doPublish();  //display incremental result
}
```

The combined power of the EDSL and progressive runtime system enables interactive visualization and analysis of extremely massive climate simulation data.



(a) Coarse resolution



(b) Full resolution

Figure 10: Pearson rank correlation between hydrophilic and hydrophobic black carbon on the 7km GEOS-5 Nature Run dataset. (a) Coarse resolution rank correlation. (b) Full-resolution rank correlation.

The two-year Non-hydrostatic 7-km Global Mesoscale Simulation "Nature Run" [9] created by NASA is one of the largest climate simulation datasets to date and an example of the future of global climate modeling. The full-resolution raw data is available in the standard NetCDF4 format, but the time and space required to download it locally seriously inhibit access and analysis. We rely on the on-demand conversion system described in Section 4.3 to handle the complexity of data loading and format conversion. The on-demand system utilizes the OPeNDAP protocol to load requested components of the massive 7km GEOS-5 "Nature Run" simulation residing at NASA. The converted multiresolution data are stored in the data cache at LLNL. The large fields of this dataset require more time to convert, but such time would have been spent downloading the data anyway. Furthermore, once converted, the fields can be accessed interactively at different resolutions by any future users.

In this example, we try to understand the relationship between hydrophilic and hydrophobic black carbon (both important environmental pollutants [38]). Hydrophobic black carbon is believed to transform into its hydrophilic sibling shortly after emission from various sources, especially industrial. In order to quickly evaluate this theory, we apply rank correlation between these two fields using remote data cached at LLNL. Each timestep of the 3d fields is approximately 1.5 GB, and our cursory analysis considered 744 timesteps, over 1.1 TB of data. Other data analysis systems would be unable to handle a volume of data this large. Yet by specifying a diminished resolution and utilizing an incremental algorithm, our script was able to start showing the results of the calculation almost immediately, and complete it for an interactively selected subregion in only a few minutes. As illustrated in Fig. 10 (a), a coarse resolution of the data was selected by the user to rapidly identify the preliminary result. The final result using the full dataset is illustrated in Fig. 10 (b).

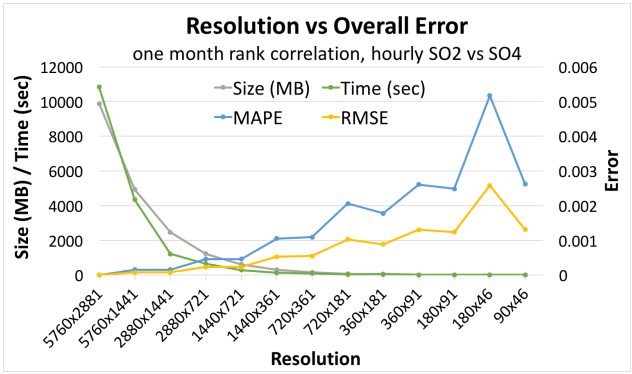


Figure 11: Comparison of data size, computation time, and root-mean-square error (RMSE) for various resolution levels in the computation of the Pearson rank correlation.

The results of computation using coarse-resolution data can be surprisingly accurate. Fig. 11 shows the root mean squared error (RMSE) between the full-resolution and several partial-resolution calculations of a 2d rank correlation within the NASA Nature Run simulation. The graph shows the relationship among error metrics, total computation size, and total computation time for each resolution level. The full-resolution computation requires nearly 100 GB of data, but at low resolutions, the error is still quite reasonable and the computation time is dramatically faster.

5.2 Combustion Simulation

Combustion simulation is crucial for modeling and analyzing complex chemical and physical processes in the search for more efficient energy utilization. One such environment is S3D [42], which has been integrated with the PIDX library [22] to directly produce multiresolution output that can be utilized by our runtime system. These simulations can produce up to terabytes of data per timestep and involve extremely large domains and hundreds of fields. However, by utilizing the multiresolution and progressive refinement, scientists can rapidly explore preliminary result instantaneously before committing to a final static analysis.

One standard analysis of combustion simulation data is to explore discrete regions of burning flame within a specific range of mixed fuel. The optional burning condition is usually achieved within

such a range. Typically, scientists will compute a derived field offline by iterating through the full-resolution volumes of both the mixture fraction and OH field and mask the OH field by mixture fraction thresholds. Despite being a simple operation, this process is computationally intensive due to the sheer size of the data. As a result, scientists often cannot repetitively experiment with different thresholds to select the best one. Compared to the commonly used workflow, our system enables scientists to interactively explore different threshold values using a simple script. Fig. 12 illustrates the process of applying the threshold. The OH field is shown in Fig. 12(a), and the masked OH field where the mixture fraction of fuel and oxygen is between 36%-40% is illustrated in Fig. 12(b). Utilizing coarse-resolution data allows very rapid cursory exploration that can be refined as necessary. The interactive exploration facilitated by this work ensures that important data is not missed nor resources wasted with unnecessary computation.

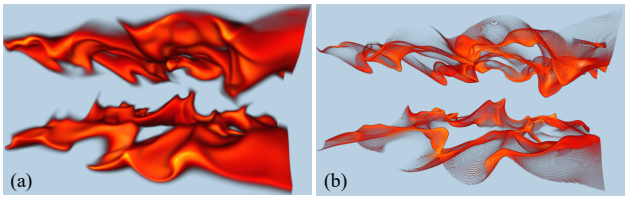


Figure 12: Exploring discrete regions of burning flame within a specific threshold of mixed fuel. (a) Shows the original OH field. (b) Shows the application of the mask to the original OH field where the mixture fraction of fuel and oxygen is between 36%-40%.

6 Discussion

In this work, we introduced a simple yet expressive embedded scripting language that abstracts the location and resolution of input data volumes, along with a runtime system to facilitate dynamic performance tuning and loop interpretations for faster convergence of incremental in-progress results. The internal data format used by the runtime system enables efficient multiresolution data loading, very fast access to regions-of-interest, multilevel architecture-independent caching, and transparent on-demand data conversion. As a whole, our system enables truly interactive analysis and visualization workflows for massive simulation ensembles, closing a gap in the existing technology. While our work is focused on structured spatiotemporal dataset, similar concepts and language extensions could be applied to unstructured data modalities in future work. We conclude this work with a discussion of the benefits as well as some of the current limitations of the system.

In order to evaluate the effectiveness of the proposed system, we had to take into account all aspects of the data analysis process, which often involves manual steps for data download and conversion and the use of multiple applications in addition to the actual computation. Because our system is intended to facilitate cursory data exploration, the primary focus has been on incrementality and interruptibility rather than optimization of any single computation. The proposed work enables many time-saving advantages over existing applications, such as transparent multiresolution data access, automatic resampling, and remote computation. To perform similar analyses using existing techniques typically requires users to manually download and resample specific variables of interest to the system that will perform the computation, and then manually construct and execute the various scripts used for the analysis. Each step can be tedious and time-consuming, and this cumbersome process curtails dynamic exploration of the analysis space. In contrast, our lightweight system enables hypotheses to be more easily tested, and even allows for more rapid discovery and validation of errors in the underlying data, as described in the *Annual Zonal Average* case study of Section 5.

The transparent data access enabled by the on-demand data reordering system provides a tremendous advantage in simplicity to enable multiresolution data access, but this system is intended as a measure to ease the transition for users of legacy data. Though computationally efficient, reordering large datasets that are not currently provided in a multiresolution format still requires time to read and write the data. We hope and anticipate that the use of reordered data formats will become more common, and our demonstration of the use of multiresolution data access to facilitate unprecedented interactive analysis of massive datasets such as the 7km NASA GEOS-5 Nature Run simulation adds to the growing body of evidence to support adoption of such formats. The settings we selected for data conversion in this work result in cached datasets that are between 17-28% larger than the original data. However, many tuning parameters can be selected for data reordering such as multiresolution bit string, block size, and methods of compression. The IDX format utilized for this work supports variable-size blocks and different data orderings within the blocks themselves (either row-major or Hz order), each of which provide trade-offs in terms of disk usage, access time, and compressibility. Data storage is an ongoing area of research, and the methods demonstrated by this work can utilize any similar type of data format to equal advantage. In other words, the proposed DSL is not tied to a specific multiresolution format.

The EDSL runtime presented in this work provides an option to perform the computation of a script using remote resources. This flexibility can be used to manually construct dataflows that utilize a combination of resources including remote servers, local systems, or GPU hardware. For future work, we would like to explore using more dynamic selection in order to make the best use of distributed computational resources. We adopted JavaScript as our host language because it was simple to write our own interpreter, but in the future, we intend to explore alternatives such as Python. Finally, we hope to extend the runtime with the ability to cache derived fields, i.e., averages, so that they could be shared like any other data in the system. Our system could then be utilized with other applications such as UV-CDAT or VisIt as a preliminary data processing facility that updates local data to be visualized by those applications.

Acknowledgments

The authors wish to thank Sasha Ames and Anthony Hoang for supporting the installation of the On-demand Data Reordering service at LLNL. This work is supported in part by NSF:CGV Award:1314896, NSF CISE ACI-0904631, NSF:IIP Award:1602127, NSF:ACI Award 1649923, DOE/Codesign P01180734, DOE/SciDAC DESC0007446, CCMSC DE-NA0002375, and PIPER: ER26142 DE-SC0010498. This material is based upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number(s) DE-NA0002375. This work is also supported in part by the Earth System Grid Federation (ESGF), the Distributed Resources for ESGF Advanced Management (DOE DREAM) project and the LLNL program on Analytics and Informatics Management Systems (AIMS). This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

References

[1] OPeNDAP Data Access Protocol, http://www.opendap.org/.

[2] J. Ahrens, B. Geveci, and C. Law. Paraview: An end-user tool for large data visualization. 01 2005.

[3] G. L. Bernstein, C. Shah, C. Lemire, Z. DeVito, M. Fisher, P. Levis, and P. Hanrahan. Ebb: A DSL for physical simluation on cpus and gpus. *CoRR*, abs/1506.07577, 2015.

[4] S. J. Camargo and A. H. Sobel. Western north pacific tropical cyclone intensity and enso. *Journal of Climate*, 18(15):2996–3006, 2005.

[5] CAPS Enterprise, Cray Inc., NVIDIA, and the Portland Group. The OpenACC Application Programming Interface v1.0, Nov 2011.

[6] Y.-J. Chiang, C. T. Silva, and W. J. Schroeder. Interactive out-of-core isosurface extraction. In *Visualization'98. Proceedings*, pages 167–174. IEEE, 1998.

[7] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, and P. Navrátil. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, pages 357–372. Oct. 2012.

[8] H. Choi, W. Choi, T. M. Quan, D. G. C. Hildebrand, H. Pfister, and W. K. Jeong. Vivaldi: A domain-specific language for volume processing and visualization on distributed heterogeneous systems. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2407–2416, Dec 2014.

[9] W. P. da Silva, A.M. and J. Nattala. File specification for the 7-km geos-5 nature run, ganymed release (non-hydrostatic 7-km global mesoscale simulation), 2014.

[10] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[11] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger. Pegasus: a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015. Funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575.

[12] J. V. der Corput. Verteilungsfunktionen. i. mitt. In *Proc. Akad. Wet. Amsterdam, 38*, pages 813–821, 1935.

[13] G. F. Diamos, A. R. Kerr, S. Yalamanchili, and N. Clark. Ocelot: A dynamic optimization framework for bulk-synchronous applications in heterogeneous systems. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, pages 353–364, New York, NY, USA, 2010. ACM.

[14] Edwards, H. C., and D. Sunderland. Kokkos array performance-portable manycore programming model. *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM 12)*, pages 1–10, 2012.

[15] J. Halton. On the efficiency of certain quasi- random sequences of points in evaluating multi- dimensional integrals, 1960.

[16] H.-C. Hege, A. Hutanu, R. Kähler, A. Merzky, T. Radke, E. Seidel, and B. Ullmer. Progressive retrieval and hierarchical visualization of large remote data. *Scalable Computing: Practice and Experience*, 6(3), 2001.

[17] N. B. J. Hoberock. *Thrust: A Productivity-Oriented Library for CUDA*, chapter 26, pages 359–371. Morgan Kaufmann, 2012.

[18] G. Kindlmann, C. Chiw, N. Seltzer, L. Samuels, and J. Reppy. Diderot: a domain-specific language for portable parallel scientific visualization and image analysis. *IEEE Transactions on Visualization and Computer Graphics (Proceedings VIS 2015)*, 22(1):867–876, Jan. 2016.

[19] F. Kjolstad, S. Kamil, J. Ragan-Kelley, D. I. W. Levin, S. Sueda, D. Chen, E. Vouga, D. M. Kaufman, G. Kanwar, W. Matusik, and S. Amarasinghe. Simit: A language for physical simulation. *ACM Trans. Graph.*, 35(2):20:1–20:21, Mar. 2016.

[20] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[21] S. Kumar, C. Christensen, J. Schmidt, P.-T. Bremer, E. Brugger, V. Vishwanath, P. Carns, H. Kolla, R. Grout, J. Chen, M. Berzins, G. Scorzelli, and V. Pascucci. Fast multiresolution reads of massive simulation datasets. In J. Kunkel, T. Ludwig, and H. Meuer, editors, *Supercomputing*, volume 8488 of *Lecture Notes in Computer Science*, pages 314–330. Springer International Publishing, 2014.

[22] S. Kumar, V. Vishwanath, P. Carns, J. Levine, R. Latham, G. Scorzelli, H. Kolla, R. Grout, R. Ross, M. Papka, J. Chen, and V. Pascucci. Efficient data restructuring and aggregation for I/O acceleration in PIDX. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 50:1–50:11. IEEE Computer Society Press, 2012.

[23] B. Langenbrunner and J. D. Neelin. Analyzing enso teleconnections in cmip models as a measure of model fidelity in simulating precipitation. *Journal of Climate*, 26(13):4431–4446, 2013.

[24] S. R. Lindemann and S. M. LaValle. Incremental low-discrepancy lattice methods for motion planning. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 2920–2927 vol.3, Sept 2003.

[25] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[26] G. Martinez, M. Gardner, and W. c. Feng. Cu2cl: A cuda-to-opencl translator for multi- and many-core architectures. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 300–307, Dec 2011.

[27] T. Maxwell. Exploratory climate data visualization and analysis using dv3d and uvcdat. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 483–487, Nov 2012.

[28] M. Mnnich and J. D. Neelin. Seasonal influence of enso on the atlantic itcz and equatorial south america. *Geophysical Research Letters*, 32(21):n/a–n/a, 2005. L21709.

[29] V. Pascucci and R. J. Frank. Global static indexing for real-time exploration of very large regular grids. In G. Johnson, editor, *Proceedings of the 2001 ACM/IEEE conference on Supercomputing, Denver, CO, USA, November 10-16, 2001, CD-ROM*, page 2. ACM, 2001.

[30] V. Pascucci, G. Scorzelli, B. Summa, P.-T. Bremer, A. Gyulassy, C. Christensen, and S. Kumar. Scalable visualization and interactive analysis using massive data streams. *Advances in Parallel Computing: Cloud Computing and Big Data*, 23:212–230, 2013.

[31] V. Pascucci, G. Scorzelli, B. Summa, P.-T. Bremer, A. Gyulassy, C. Christensen, S. Philip, and S. Kumar. *The ViSUS Visualization Framework*, chapter 19, pages 401–414. Chapman \& Hall/CRC Computational Science, 2012.

[32] e. a. R. Hornung, J. Keasler. The raja portability layer: Overview and status, 2014.

[33] P. Rautek, S. Bruckner, M. E. Grller, and M. Hadwiger. Vislang: A system for interpreted domain-specific languages for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2388–2396, Dec 2014.

[34] R. Rew and G. Davis. Netcdf: an interface for scientific data access. *IEEE Computer Graphics and Applications*, 10(4):76–82, July 1990.

[35] E. Santos, J. Poco, Y. Wei, S. Liu, B. Cook, D. N. Williams, and C. T. Silva. UV-CDAT: Analyzing climate datasets from a user's perspective. *Computing in Science and Engineering*, 15(1):94–103, Jan./Feb. 2013.

[36] K. E. Taylor, R. J. Stouffer, and G. A. Meehl. An overview of cmip5 and the experiment design. *Bulletin of the American Meteorological Society*, 93(4):485–498, 2012.

[37] Y. Tian, S. Klasky, W. Yu, B. Wang, H. Abbasi, N. Podhorszki, and R. Grout. Dynam: Dynamic multiresolution data representation for large-scale scientific analysis. In *Networking, Architecture and Storage (NAS), 2013 IEEE Eighth International Conference on*, pages 115–124, July 2013.

[38] United States Environmental Protection Agency. Report to congress on black carbon. In *Government Printing Office, EPA-450/R-12-001*, Mar 2012.

[39] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[40] A. B. P. Welford and B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, pages 419–420, 1962.

[41] K. Wu. Fastbit: an efficient indexing technology for accelerating data-intensive science. 16(1):556, 2005.

[42] C. S. Yoo, R. Sankaran, and J. H. Chen. Three-dimensional direct numerical simulation of a turbulent lifted hydrogen jet flame in heated coflow: flame stabilization and structure. *Journal of Fluid Mechanics*, pages 453–481, 2009.