Chapter 19

The ViSUS Visualization Framework

V. Pascucci, G. Scorzelli, B. Summa, P.-T. Bremer
University of Utah

A. Gyulassy, C. Christensen, S. Philip, S. Kumar
University of Utah

The ViSUS software framework[1] has been designed as an environment that allows the interactive exploration of massive scientific models on a variety of hardware, possibly over platforms distributed geographically. This chapter is devoted to the description of the scalability principles that are at the basis of the ViSUS design and how they can be used in practical applications, both in scientific visualization and other domains such as digital photography or exploration of geospatial models.

19.1 Introduction

The ViSUS software framework was designed with the primary philosophy that the visualization of massive data need not be tied to specialized hardware or infrastructure. In other words, a visualization environment for large data can be designed to be lightweight, highly scalable and run on a variety of platforms or hardware. Moreover, if designed generally such an infrastructure can have a wide variety of applications, all from the same code base. Figure 19.1 details example applications and the major components of the ViSUS infrastructure. The components can be grouped into three major categories. First, a lightweight and fast out-of-core data management framework using multi-resolution space filling curves. This allows the organization of information in an order that exploits the cache hierarchies of any modern data storage architectures. Second, a dataflow framework that allows data to be processed

[1]For more information and software downloads see `http://visus.co` and `http://visus.us`

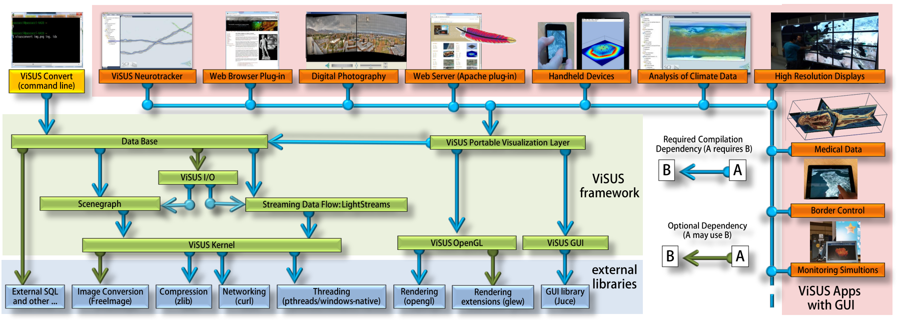

FIGURE 19.1: The ViSUS software framework. Arrows denote external and internal dependences of the main software components. Additionally we show the relationship with several example applications that have been successfully developed with this framework.

during movement. Processing massive datasets in their entirety would be a long and expensive operation which hinders interactive exploration. By designing new algorithms to fit within this framework, data can be processed as it moves. Third, a portable visualization layer which was designed to scale from mobile devices to powerwall displays with same code base. In this chapter we will describe the ViSUS infrastructure, as well as give practical examples of its use in real world applications.

19.2 ViSUS Software Architecture

Figure 19.1 provides a diagram of the ViSUS software architecture. In this section we will detail ViSUS's three major components and how they are used to achieve a fast, scalable, and highly portable data processing and visualization environment.

Data Access Layer The ViSUS data access layer is a key component allowing immediate, efficient data pipeline processing that otherwise would be stalled by traditional system I/O cost. In particular, the ViSUS I/O component (and its generalized Data Base component) are focused on enabling the effective deployment of Out-of-Core and data streaming algorithms. Out-of-core computing [11] specifically addresses the issues of algorithm redesign and data layout restructuring. These are necessary to enable data access patterns having minimal performance degradation with external memory storage. Algorithmic approaches in this area also yield valuable techniques for parallel and distributed computing. In this environment, one typically has to deal with the similar issue of balancing processing time with the time required for data access and movement amongst elements of a distributed or parallel application.

The solution to the out-of-core processing problem is typically divided into two parts: (1) algorithm analysis, to understand data access patterns and, when possible, redesign to maximize data locality; (2) storage of data in secondary memory using a layout consistent with the access patterns of the algorithm, amortizing the cost of individual I/O operations over several memory access operations.

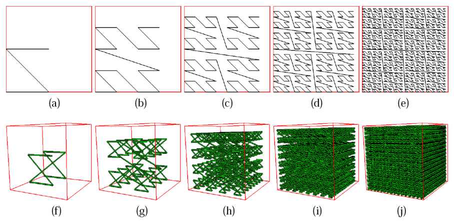

FIGURE 19.2: (A-E) The first five levels of resolution of the 2D Lebesgue's space filling curve. (F-J) The first five levels of resolution of the 3D Lebesgue's space filling curve.

To achieve real-time rates for visualization and/or analysis of extreme scale data, one would commonly seek some form of adaptive level of detail and/or data streaming. By traversing simulation data hierarchically from the coarse to the fine resolutions and progressively updating output data structures derived from this data, one can provide a framework that allows for real-time access of the simulation data that will perform well even on extreme scale data. Many of the parameters for interaction, such as display viewpoint, are determined by users at run time and therefore precomputing these levels of details optimized for specific queries is infeasible. Therefore to maintain efficiency, a storage data layout must satisfy two general requirements: (i) if the input hierarchy is traversed in coarse-to-fine order, data in the same level of resolution should be accessed at the same time, and (ii) within each level of resolution, the regions in close spatial proximity are stored in close proximity in memory.

Space filling curves [9] have been used successfully to develop a static indexing scheme that generates a data layout satisfying both the above requirements for hierarchical traversal, see Figure 19.2. The data access layer of ViSUS employs a hierarchical variant of a Lebesgue space filling curve [5]. The data layout of this curve is commonly referred to as *HZ order* in the literature. This data access layer has three key features that make it particularly attractive. First, the order of the data is independent of the out-of-core block structure, so that its use in different settings (e.g. local disk access or transmission over a network) does not require any large data reorganization. Second, conversion from the Z-order indexing [4] used in classical database approaches to ViSUS's HZ-order indexing scheme can be implemented with a simple sequence of bit-string manipulations. Third, since there is no data replication, we avoid the performance penalties associated with guaranteeing consistency especially for dynamic updates, as well as increased storage requirements typically associated with most hierarchical and out-of-core schemes.

Parallel I/O for Large Scale Simulations The multi-resolution data layout of ViSUS discussed above is a progressive, linear format and therefore has a write routine that is inherently serial. During the execution of large scale simulations, it would be ideal for each node in the simulation to be able to write its piece of the domain data directly into this layout. Therefore a parallel write strategy must be employed. Figure 19.3 illustrates different possible parallel strategies that have been considered. As shown in Figure 19.3 (A), each process can naively write its own data directly to the proper location in a unique underlying binary file. This is inefficient due to the large number of small granularity, concurrent accesses to the same file. Moreover, as the data gets large, it becomes disadvantageous to store the entire dataset as a single, large file and typically the entire dataset is partitioned into a series of smaller more manageable pieces. This disjointness can be used by a parallel write routine. As each simulation process produces a portion of the data, it can store its piece of the overall dataset locally and pass the data on to an aggregator process.

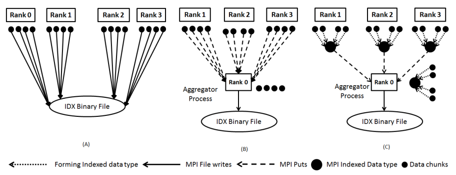

FIGURE 19.3: Parallel I/O strategies: (a) Naive approach where each process writes its data in the same file, (a) alternative approach where contiguous data segment to an intermediate aggregator that writes to disk, (c) communication reducing approach with bundeling of noncontiguous accesses into a single message.

These aggregator processes can be used to gather the individual pieces and composite the entire dataset. Figure 19.3 (B) shows this strategy, where each process transmits a contiguous data segment to an intermediate aggregator. Once the aggregator's buffer is complete, the data is written to disk using a single large I/O operation. Figure 19.3 (C), illustrates a strategy where several noncontiguous memory accesses from each process are bundled into a single message. This approach also reduces the overhead due to the number of small network messages needed to transfer the data to the aggregators. This strategy has been shown to exhibit good throughput performance and weak scaling for S3D combustion simulation applications when compared to standard Fortran I/O benchmark [2, 3]. In particular, recent results[2] have shown empirically how this strategy scales well for a large number of nodes (currently up to 32,000) while enabling real-time monitoring of high resolution simulations (see Section 19.3).

LightStream Dataflow and Scene Graph. Even simple manipulations can be overly expensive when applied to each variable in a large scale dataset. Instead, it would be ideal to process the data based on need by pushing data through a processing pipeline as the user interacts with different portions of

[2]Execution on the Hopper 2 system at NERSC.

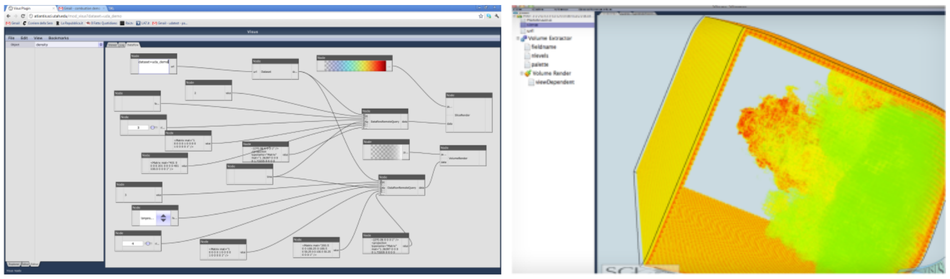

FIGURE 19.4: The LightStream Dataflow used for analysis and visualization of a 3D combustion simulation (Uintah code). (left) Several dataflow modules chained together to provide a light and flexible stream processing capability. (right) One visualization that is the result from this dataflow.

the data. The ViSUS multi-resolution data layout enables efficient access to different regions of the data at varying resolutions. Therefore different compute modules can be implemented using progressive algorithms to operate on this data stream. Operations such as binning, clustering, or rescaling are trivial to implement on this hierarchy given some known statistics on the data, such as the function value range, etc. These operators can be applied to the data stream as-is, while the data is moving to the user, progressively refining the operation as more data arrives. More complex operations can also be reformulated to work well using the hierarchy. For instance, using the layout for 2-dimensional image data produces a hierarchy which is identical to a sub-sampled image pyramid on the data. Moreover as data is requested progressively, the transfer will traverse this pyramid in a coarse-to-fine manner. Techniques such as gradient-domain image editing can be reformulated to use this progressive stream and produce visually acceptable solutions [10, 7, 8]. These adaptive, progressive solutions allow the user to explore a full resolution solution as if it was fully available, without the expensive, full computation.

ViSUS LightStream facilitates this steam processing model by providing definable modules within a dataflow framework with a well understood API. Figure 19.4 gives an example of a dataflow for the analysis and visualization of a scientific simulation. This particular example is the dataflow for a Uintah combustion simulation used by the Center for the Simulation of Accidental Fires and Explosions (C-SAFE) at the University of Utah. Each LightStream module provides streaming capability through input and output data ports that can be used in a variety of data transfer/sharing modes. In this way, groups of modules can be chained to provide complex processing operations as the data is transferred from the initial source to the final data analysis and visualization stages. This data flow is typically driven by user demands and interactions. A variety of "standard" modules, such as data differencing (for change detection), content based image clustering (for feature detection), or volume rendering with multiple, science-centric transfer functions, are part of

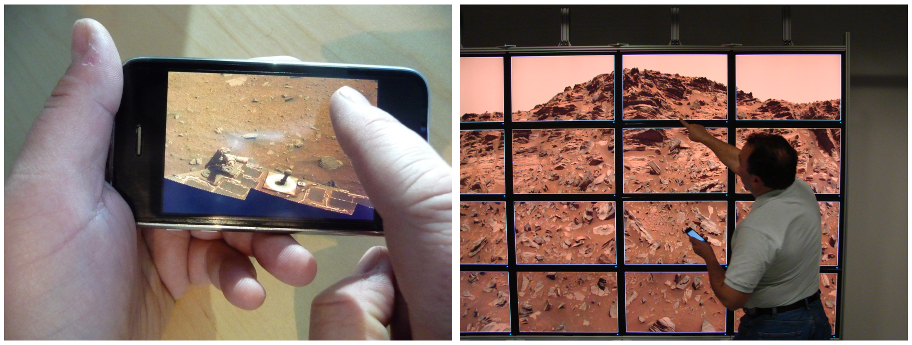

FIGURE 19.5: The same application and visualization of a Mars panorama running on an iPhone 3G mobile device (left) and a powerwall display (right). Data courtesy of NASA.

the base system. These can be used by new developers as templates for their own progressive streaming data processing modules.

ViSUS also provides a scene graph hierarchy for both organizing objects in a particular environment, as well as the sharing and inheriting of parameters. Each component in a model is represented by a node in this scene graph and inherits the transformations and environment parameters from its parents. 3D volume or 2D slice extractors are children of a data set node. As an example of inheritance, a scene graph parameter for a transfer function can be applied to the scene graph node of a data set. If the extractor on this data set does not provide its own transfer function, it will be inherited.

Portable Visualization Layer - ViSUS AppKit. The visualization component of ViSUS was built with the philosophy that a single code base can be designed to run on a variety of platforms and hardware ranging from mobile devices to powerwall displays. To enable this portability, the the basic draw routines were designed to be OpenGL ES compatible. This is a limited subset of OpenGL used primarily for mobile devices. More advanced draw routines can be enabled if a system's hardware can support it. In this way, the data visualization can scale in quality depending on the available hardware. Beyond the display of the data, the underlying GUI library can hinder portability to multiple devices. At this time ViSUS has made use of the Juce [3] library which is lightweight and supports mobile platforms such as iOS and Android in addition to major operating systems. ViSUS provides a demo viewer which contains standard visualizations such as slicing, volume rendering and isosurfacing. Similarly to the example LightStream modules, these routines can be expanded through a well-defined API. Additionally, the base system can display 2D and 3D time- varying data. As mentioned above, each of these visualizations can operate on the end result of a LightStream dataflow. The

[3]http://www.rawmaterialsoftware.com

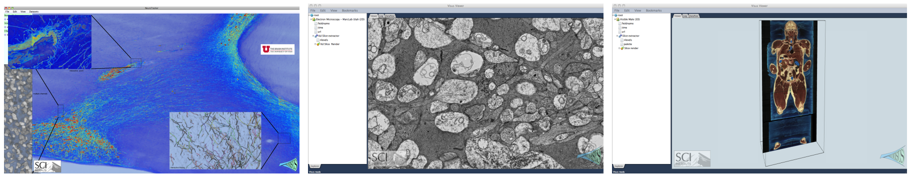

FIGURE 19.6: The ViSUS software framework visualizing and processing medical imagrey. (left) The Neurotracker application providing the segmentation of neurons from extremely high resolution Confocal Fluorescence Microscopy brain imagery. This data courtesy of the Center for Integrated Neuroscience and Human Behavior at the Brain Institute, University of Utah. (middle) An application for the interactive exploration of an electron microscopy image of a slice of a rabbit retina. This dataset is courtesy of the MarcLab at the University of Utah. (right) A 3D slicing example using the Visible Male dataset.

system considers a 2D dataset as a special case of a slice renderer and therefore the same code base is used for 2D and 3D datasets. Combining all of the above design decisions allows the same code base to be used on multiple platforms seamlessly for data of arbitrary dimensions. Figure 19.5 shows the same application and visualization running on an iPhone 3G mobile device and a powerwall display.

19.3 Applications

As shown in the upper and right portions of the infrastructure diagram in Figure 19.1, ViSUS has the versatility to be used in a wide range of applications. Below we will highlight a representative subset of these applications. The general philosophy behind ViSUS applications is the deployment of light tools that are task driven as a complement to general purpose solutions provided by other existing systems.

Neurotracker and Other Medical Applications. The Neurotracker is an application built on the ViSUS framework that targets the segmentation of neurons from extremely high resolution Confocal Fluorescence Microscopy brain imagery [4]. The core data processing of the Neurotracker uses the ViSUS I/O library to get fast access to the brain imaging data combined with multiresolution topological analysis used to seed the segmentation of neurons. A marching image segmentation routine extracts filament structures from the

[4]Data is courtesy of the Center for Integrated Neuroscience and Human Behavior at the Brain Institute, University of Utah (http://brain.utah.edu/)

topological seeds. The user interface of the the Neurotracker is built with the ViSUS GUI components specialized for segmentation.

Figure 19.6 also highlights two additional medical imaging applications. In Figure 19.6 (center), an example of the ViSUS framework is used for interactive exploration of an electron microscopy image of a slice of a rabbit retina. In all, this 2D dataset is over 3.4 gigapixels in size[5]. Figure 19.6 (right) is a 3D data slicing example for Visible Male[6] dataset comprised of over 4.6 billion color voxels.

Web-server and plug-in ViSUS has been extended to support a client-server model in addition to the traditional viewer. The ViSUS server can be used as a standalone application or a web server plugin module. The ViSUS server uses HTTP (a stateless protocol) in order to support many clients. A traditional client/server infrastructure, where the client established and maintained a stable connection to the server, can only handle a limited number of clients robustly. Using HTTP, the ViSUS server can scale to thousands of connections. The Visus client keeps a number (normally 48) of connections alive in a pool using the "keep-alive" option of HTTP. The use of lossy or lossless compression is configurable by the user. For example, ViSUS supports JPEG and EXR for lossy compression of byte and float data respectively. The ViSUS server is an open client/server architecture, therefore it is possible to port the plugin to any web server which supports a C++ module (i.e. apache, IIS). The ViSUS client can be enabled to cache data to local memory or to disk. In this way, a client can minimize transfer time by referencing data already sent, as well as having the ability to work offline if the server becomes unreachable. The ViSUS portable visualization framework (Appkit) also has the ability to be compiled as a Google Chrome, Microsoft Internet Explorer, or Mozilla Firefox web browser plugin. This allows a ViSUS framework based viewer to be easily integrated into web visualization portals.

Remote Climate Analysis and Visualization The ViSUS software framework has been used in the climate modeling community to visualize climate change simulations comprised of many species over a large number of time steps. This type of data provides the opportunity to both build high quality data analysis routines and challenge the performance of the data management infrastructure. Figure 19.7, shows ViSUS rendering 10TB of data used by a finalist in the data transfer challenge of Supercomputing 2009. This work showed the possibility of transferring, transforming, analyzing, and rendering a large dataset on geographically distributed computing resources [1]. At the same time, this work involved was used to introduce analysis and visualizations providing novel insights into the dynamics of global carbon cycle, atmospheric chemistry, land and ocean ecological processes and their coupling

[5]Data is courtesy of the MarcLab at the University of Utah (http://prometheus.med.utah.edu/~marclab/)

[6]http://www.nlm.nih.gov/research/visible/visible_human.html

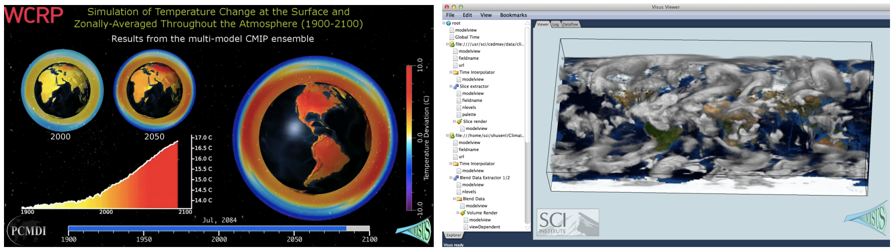
WCRP
Simulation of Temperature Change at the Surface and Zonally-Averaged Throughout the Atmosphere (1900-2100)
Results from the multi-model CMIP ensemble
2000
2050
Temperature Deviation (C)
Jul, 2084
PCMDI
1900
1950
2000
2050
2100
SCI

FIGURE 19.7: Remote climate visualization with ViSUS. (left) The ViSUS framework providing a visualization for a temperature change ensemble simulation for the Earth's surface for the December 2009 climate summit meeting in Copenhagen. (right) A visualization of global cloud density for a more recent climate simulation.

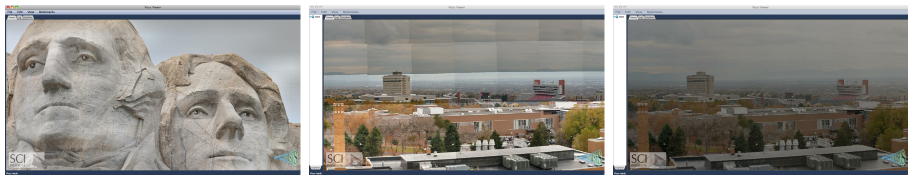

FIGURE 19.8: (left) A visualization of a 2D panorama dataset of Mount Rushmore (500 megapixel). Dataset courtesy of City Escapes Photography. (middle) The color shift between images in a panorama mosaic. (right) An application using a Lightstream dataflow to provide approximate gradient domain solution as a user interacts with the data.

with climate. In particular, this work was also used to present recent findings regarding the Earth's temperature change based on historical and projected simulation data at the December 2009 climate summit meeting in Copenhagen (Denmark). In Figure 19.7 (right), we see the ViSUS framework providing a visualization of global cloud density for a more recent simulation.

Panorama Multiscale Processing and Viewer The ViSUS framework along with the Lightstream Dataflows can be used for real-time, large panorama processing and visualization. In Figure 19.8 (left) we have the visualization of a 500 megapixel 2D image of Mount Rushmore[7]. In this fattened image, you can detect color shifts due the fact that it is a mosaic of many individual images. Figure 19.8 (middle) provides a visualization of the original picture data for a panorama of Salt lake City, which is comprised of over 600 individual images for a total image mosaic size of 3.2 gigapixel. As mentioned in Section 19.2, the Lightstream Dataflow can be used to operate on the multi-resolution data as the panorama is being viewed and provide an approximate gradient domain solution [10, 7, 8] for each viewpoint. In this way, a user can explore the panorama as if the full gradient domain solution

[7]Data is courtesy of City Escapes Photography (http://www.cityescapesphotography.com)

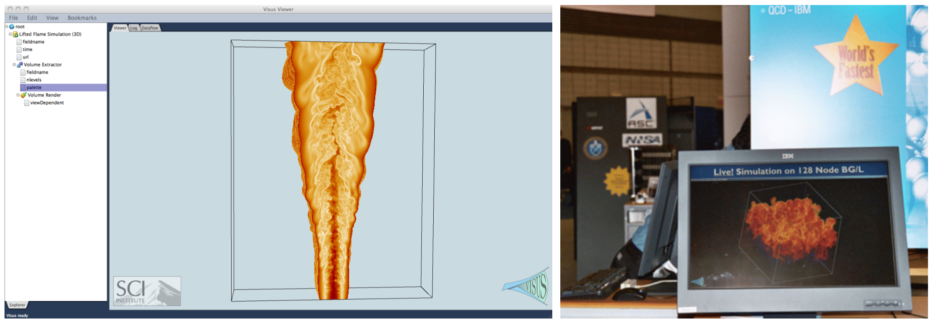

FIGURE 19.9: Remote visualization and monitoring of simulations. (left) An S3D combustion simulation visualized from a desktop in the SCI institute (Slat Lake City, Utah) during its execution on the HOPPER 2 high performance computing platform in Lawrence Berkeley National Laboratory (Berkeley, California). (right) Two ViSUS demonstrations of LLNL simulation codes (Miranda and Raptor) visualized in real-time while executed on the BlueGene/L prototype installed at the IBM booth of the Supercomputing exhibit.

was available without it ever being computed in full. This preview is shown in Figure 19.8 (right).

Real-Time Simulation Monitoring Ideally, a user-scientist would like to view a simulation as it is computed, in order to steer or correct the simulation as unforeseen events arise. Simulation data is often very large. For instance, a single field of a time-step from the S3D combustion simulation in Figure 19.9 (left) is approximately 128 GB in size. In the time needed to transfer this single time-step, the user-scientist would have lost any chance for significant steering/correction of an ongoing simulation or at least take the opportunity to save computing resources by early termination of a job that is not useful anymore. By using the parallel ViSUS data format in simulation checkpointing [2, 3], we can link this data directly with an Apache server using a ViSUS plug-in running on a node of the cluster system. By doing this, user-scientists can visualize simulation data as checkpoints are reached. ViSUS can handle missing or partial data, therefore the data can be visualized even as it is being written to disk by the system.

ViSUS's support for a wide-variety of clients (a stand-alone application, a web-browser plug-in, or an iOS application for the iPad or iPhone) allows the application scientist to monitor a simulation as it is produced, on practically any system that is available without any need to transfer the data off the computing cluster. As mentioned above, Figure 19.9 (left) is an S3D large-scale, combustion simulation visualized remotely from an HPC platform[8].

This work is the natural evolution of the ViSUS approach of targeting

[8]Data is courtesy of Jackie Chen at Sandia National Laboratories, Combustion Research Facility `http://ascr.sandia.gov/people/Chen.htm`

practical applications for out-of-core data analysis and visualization. This approach has been used for direct streaming and real-time remote monitoring of the early large scale simulations such as those executed on the IBM BG/L supercomputers at LLNL [6] shown in Figure 19.9 (right). This work continues its evolution towards the deployment of high performance tools for in-situ and post-processing data management and analysis for the software and hardware resources of the future including exascale DOE platforms of the next decade[9].

Bibliography

[1] Rajkumar Kettimuthu and Others. Lessons learned from moving earth system grid data sets over a 20 gbps wide-area network. In *SC*, pages 194–198. ACM, Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC 2010).

[2] S. Kumar, V. Pascucci, V. Vishwanath, P. Carns, R. Latham, T. Peterka, M. Papka, and R. Ross. Towards parallel access of multi-dimensional, multiresolution scientific data. In *Proceedings of 2010 Petascale Data Storage Workshop*, November 2010.

[3] S. Kumar, V. Vishwanath, P. Carns, B. Summa, G. Scorzelli, V. Pascucci, R. Ross, J. Chen, H. Kolla, and R. Grout. Pidx: Efficient parallel i/o for multi-resolution multi-dimensional scientific datasets. In *Proceedings of IEEE Cluster 2011*, September 2011.

[4] J. K. Lawder and P. J. H. King. Using space-filling curves for multidimensional indexing. *Lecture Notes in Computer Science*, 1832:20, 2000.

[5] Valerio Pascucci and Randall J. Frank. Global static indexing for realtime exploration of very large regular grids. In *SC*, page 2, 2001.

[6] Valerio Pascucci, Daniel E. Laney, Ray J. Frank, F. Gygi, Giorgio Scorzelli, Lars Linsen, and Bernd Hamann. Real-time monitoring of large scientific simulations. In *SAC*, pages 194–198. ACM, 2003.

[7] Sujin Philip, Brian Summa, Peer-Timo Bremer, and Valerio Pascucci. Parallel Gradient Domain Processing of Massive Images. In Torsten Kuhlen, Renato Pajarola, and Kun Zhou, editors, *Eurographics Symposium on Parallel Graphics and Visualization*, pages 11–19, Llandudno, Wales, UK, 2011. Eurographics Association.

[9]Center for Exascale Simulation of Combustion in Turbulence (ExaCT) `http://science.energy.gov/ascr/research/scidac/co-design/`

[8] Sujin Philip, Brian Summa, Valerio Pascucci, and Peer-Timo Bremer. Hybrid cpu-gpu solver for gradient domain processing of massive images. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 244 –251, dec. 2011.

[9] Hans Sagan. *Space-Filling Curves*. Springer-Verlag, New York, NY, 1994.

[10] B. Summa, G. Scorzelli, M. Jiang, P.-T. Bremer, and V. Pascucci. Interactive editing of massive imagery made simple: Turning atlanta into atlantis. *ACM Trans. Graph.*, 30:7:1–7:13, April 2011.

[11] J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, March 2000.