Fast Multiresolution Reads of Massive Simulation Datasets

Sidharth Kumar[1], Cameron Christensen[1], John A. Schmidt[1], Peer-Timo Bremer[1,4], Eric Brugger[4], Venkatram Vishwanath[2], Philip Carns[2], Hemanth Kolla[3], Ray Grout[5], Jacqueline Chen[3], Martin Berzins[1], Giorgio Scorzelli[1], and Valerio Pascucci[1]

[1] SCI Institute, University of Utah, Salt Lake City, UT, USA
[2] Argonne National Laboratory, Argonne, IL, USA
[3] Sandia National Laboratory, Livermore, CA, USA
[4] Lawrence-Livermore National Laboratory, Livermore, CA, USA
[5] National Renewable Energy Laboratory, Golden, CO, USA
sidharth.kumar@utah.edu

Abstract. Today's massively parallel simulation codes can produce output ranging up to many terabytes of data. Utilizing this data to support scientific inquiry requires analysis and visualization, yet the sheer size of the data makes it cumbersome or impossible to read without computational resources similar to the original simulation. We identify two broad classes of problems for reading data and present effective solutions for both. The first class of data reads depends on user requirements and available resources. Tasks such as visualization and user-guided analysis may be accomplished using only a subset of variables with a restricted spatial extent at a reduced resolution. The other class of reads requires full resolution multivariate data to be loaded, for example to restart a simulation. We show that utilizing the hierarchical multiresolution IDX data format enables scalable and efficient serial and parallel read access on a variety of hardware from supercomputers down to portable devices. We demonstrate interactive view-dependent visualization and analysis of massive scientific datasets using low-power commodity hardware, and we compare read performance with other parallel file formats for both full and partial resolution data.

Keywords: parallel I/O, multiresolution, PIDX, read performance, interactive visualization, S3D, Uintah, VisIt

1 Introduction

Massively parallel scientific simulations often generate large datasets that can range in size up to many terabytes. Reading this data is required for several reasons. Due to crashes, code modifications or limited job time, simulations must often be restarted from an intermediate timestep. Restarts require reading of the entire saved state of the simulation at full resolution for a given time. Similar to simulation restarts, comprehensive postprocessing analysis also requires reading

of full resolution data, athough only a subset of variables may be necessary. The other major reasons to read data are for user-directed analysis and visualization. Unlike restarts and postprocessing, these tasks can often be performed using lower-resolution data with less spatial extent. For example, coarse resolution data can be used to compute an approximation of comprehensive analysis results. Similarly, the limited size of display devices permits lower-resolution data to be shown with no perceptible difference in visual quality. Finally, spatial extent can be restricted since only data within a visible region actually needs to be loaded. Thus, we identify two general classes of data reading for large scientific data: (a) full resolution reads of an entire dataset as required for simulation restarts, and (b) partial resolution reads of a subset of the data suitable for visualization and cursory analysis tasks. Complete reads are generally performed using parallel systems while partial reads may be done in serial or parallel.

We describe three improvements for reading massively parallel simulation checkpoint data at both full and partial resolutions. First, we demonstrate the utility of coarse resolution reads using spatially restricted subsets of the domain for view-dependent rendering in which only data within the field of view at the granularity necessary to support the display device needs to be loaded. Spatially restricted coarse resolution reads facilitate faster data loading as well as the ability to visualize massive simulation data using modest commodity hardware on which it would have previously been impossible to load full datasets. We provide a plugin and infrastructural modifications to enable view-dependent rendering using the VisIt visualization tool, a distributed parallel application currently in use on many supercomputers. Next, we demonstrate fast full resolution file loads of complete checkpoint data for parallel simulation restarts or comprehensive analysis. Finally, we evaluate parallel reading of data at varying resolutions suitable for summary analysis and visualization. Our work includes three significant contributions for reading massive simulation data:

- interactive view-dependent visualization of massive datasets using VisIt
- efficient parallel reads of complete checkpoint data for simulation restarts
- parallel multiresolution reads for summary analysis

In order to address both parallel full resolution reads, as well as parallel or serial partial resolution reads we utilize the IDX data format [20]. IDX is a hierarchical multiresolution data format with support for both lightweight serial and fast distributed parallel I/O. Parallel I/O is performed using PIDX[15], a fast parallel library for reading and writing IDX multiresolution data.

The remaining sections of this paper are organized as follows. Section 2 explains relevant background work involving the IDX multiresolution file format, view-dependent rendering and the VisIt visualization and analysis application. Section 3 explains the setup of our experiments and the integration of the PIDX I/O library with the Uintah simulation code. Sections 4-6 discuss our experimental results with view-dependent serial reads, full resolution parallel reads and multiresolution parallel reads. We discuss related work in Section 7 and conclude in Section 8.

2 Background

One of the important observations of our work is that utilizing a hierarchical multiresolution data format facilitates faster and less expensive visualization with no apparent reduction in quality. For our experiments we utilize the IDX format. We briefly introduce this format below as well as the PIDX library for parallel I/O. Next, we discuss the mechanism of view-dependent rendering. Finally, we describe the VisIt parallel distributed visualization system for which we have integrated support for reading IDX data.

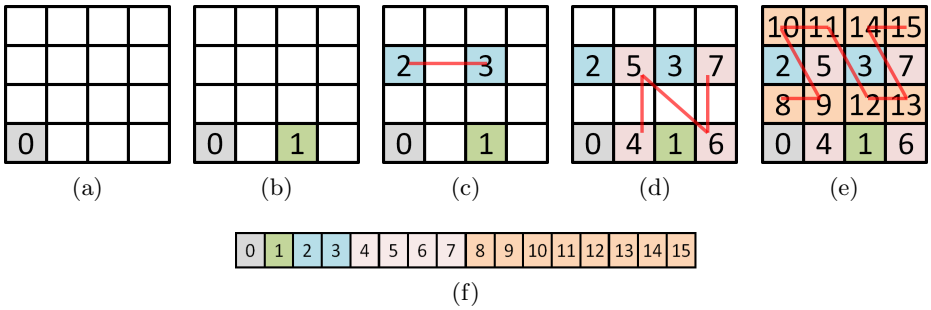


Fig. 1: HZ ordering (indicated by box labels and curves) at various levels for a 4x4 dataset. (a)-(e) Levels 0-4, respectively. (f) File level arrangement of data points in the IDX format.

2.1 IDX Multiresolution File Format

IDX is a multiresolution file format that enables fast and efficient access to large-scale scientific data. The format provides efficient, cache oblivious, progressive access to data by utilizing a hierarchical Z (HZ) order for storage [20]. The HZ order is calculated for each data sample using the spatial coordinates of that sample. The samples are then assigned a particular HZ level, analogous to the different levels of resolution at which data can be retrieved, from coarse to fine (see figure 1(a-e)). With each increasing level, the number of elements increases by a factor of 2. Data in an IDX file is written with an increasing HZ order (see figure 1(f)).

The data access layer of IDX format has three salient 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 data reorganization. This property is especially important in a parallel setting to facilitate restarts with varying numbers of nodes and reading data that originally might have been produced by a different number of nodes. Second, conversion from the Z-order

indexing [16] used in classical database approaches to the IDX HZ-order indexing scheme can be implemented with a simple sequence of bit-string manipulations. Finally, since there is no redundant data stored for different resolutions, IDX does not incur the increased storage requirements associated with most other hierarchical and out-of-core data management schemes.

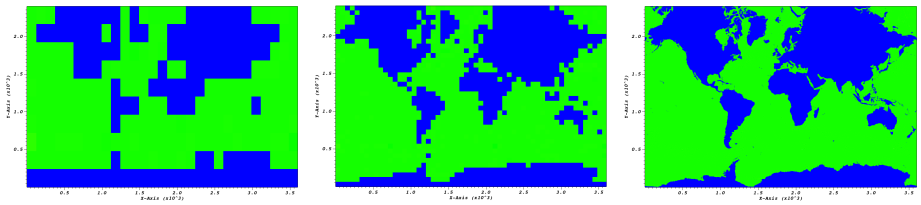
2.2 PIDX: Parallel IDX

IDX is a desirable file format for visualization of large-scale simulation results because of its ability to access multiple levels of resolution with low latency for interactive exploration. The Parallel IDX (PIDX) library enables parallel simulations to directly write IDX data [13]. The library coordinates data access among processes to concurrently write to the same dataset with coherent results.

Our previous work has been dedicated to the design and optimization of PIDX for writing data in the IDX format, as opposed to this work that focuses on reads. In [13], we demonstrated the use of PIDX in coordinating parallel write access to multidimensional, regular datasets, and explored several low-level encoding and file access optimizations. In subsequent work [15], we introduced a two-phase I/O strategy [9] in which each process first performs a local HZ encoding of its own data. The data is then aggregated by a subset of processes (aggregators) that in turn write it to disk. This strategy avoids suboptimal small-sized file accesses from each process in lieu of large, contiguous accesses performed by a select few aggregators.

2.3 View-Dependent Rendering

View-dependent rendering is a combination of techniques designed to facilitate interactive graphical applications consisting of frustum-based scene culling and level-of-detail based geometry and texture loading. Scene culling, commonly known as "frustum clipping," is an integral part of the OpenGL and DirectX rasterization pipelines [3] and most types of interactive applications from games to CAD systems rely on this technique for fast drawing [2]. The notion is simple: data outside the viewing frustum need not be considered for shading a given scene. One example of level-of-detail texture loading is MIP mapping, introduced by [27], which automatically determines the optimal resolution texture data to be used when rendering a scene from a given viewpoint based on the distance of the texture to the viewer. These maps are typically precomputed from full-resolution textures and stored on disk as an image "pyramid" where each successive level is half the size of the previous level, effectively doubling the size of the original image. MIP mapping has two major advantage: (a) it avoids loading unnecessary texture data into limited graphics memory, and (b) it permits faster rasterization since textures need not be sampled more than necessary. However, for extremely large textures or 3D volumes, computing and storing these image pyramids is generally too cumbersome.



(a) (b) (c)

Fig. 2: Coarse-to-fine resolution (HZ) levels of a 2D dataset shown in VisIt.

2.4 VisIt Visualization Application

VisIt [5] is a popular distributed parallel visualization and analysis application used by many scientists. VisIt is typically executed in parallel where it coordinates visualization and analysis tasks for massive simulation data. The data is typically read at its full resolution, requiring as much system memory as the size of the data being read to perform its work. In order to leverage the multiresolution capability of the IDX format, we have added a plugin to read IDX data and incorporated dynamic view-dependent read capabilities with VisIt as well as enabling visualization of IDX data hosted on remote servers. With these additions it is now possible to perform view-dependent visualization and analysis using a tiny fraction of the compute power previously required for these tasks. Manual control of the resolution level is also permitted. Figure 2 shows an IDX dataset in VisIt at different resolution levels.

2.5 Related Work

Parallel I/O file formats have been developed to aid in structuring data to facilitate efficient data storage and access. Some of the popular parallel file formats are Parallel HDF5 [1], Parallel NetCDF [17], MPI-IO [24], VAPOR [4] and ADIOS [18].

Multiresolution data formats for parallel processing environments have been studied previously in [11, 25, 6]. Chiueh leverages Gaussian and Laplacian Pyramid transforms tailored according to the underlying storage architecture to improve read performance [8]. Chaoli, Jinzhu and Han have presented work in parallel visualization [26] of multiresolution data. Their algorithm involves conversion of raw data to a multiresolution wavelet tree and focuses more on parallel visualization rather than a generic data format such as PIDX. More recently, DynaM data representation supports convolution-based multiresolution data representation [25]. Ahrens et. al. work enables multiresolution visualization by sampling existing data, and writes the multiresolution levels and meta-data to disk as independent files while keeping the full-resolution file intact [6].

The ViSUS Viewer [22, 21] is an application that facilitates online visualization and analysis. It has been designed to allow interactive exploration

of massive scientific models on a variety of hardware from desktops down to hand-held devices. ViSUS supports thread-parallel operation in contrast to the distributed parallel mechanism provided by VisIt. The ViSUS Viewer natively supports the IDX streaming data format.

3 Experiment Setup

The experiments presented in this work were performed on Edison at the National Energy Research Scientific Computing (NERSC) Center and Tukey at the Argonne Leadership Computing Facility (ALCF). Edison is a Cray XC30 with a peak performance of 2.39 petaflops, 124, 608 compute cores, 332 TiB of RAM and 7.5 PiB of online disk storage. We used Edison Lustre file system (168 GiB/s, 24 I/O servers and 4 Object Storage Targets). Tukey is a visualization cluster consisting of 96 compute nodes. Each node consists of two 2GHz AMD Opteron 6128 8 core CPUs (16 cores per node and 64 GB RAM) and two NVIDIA Tesla M2070 GPUs (6 GB GPU RAM). All nodes are connected via QDR Infiniband interconnect and share the same GPFS filesystem with Mira (a 768K core BlueGene/Q system). We used one of the /project scratch filesystems for doing the visualization read experiments.

S3D Simulation Code: S3D is a continuum scale first principles direct numerical simulation code that solves the compressible governing equations of mass continuity, momenta, energy and mass fractions of chemical species including chemical reactions. The computational approach in S3D is described in [7]. Each run of S3D generates four variables: pressure, temperature, velocity (3 samples) and species (11 samples). Details of integration of the PIDX I/O library with S3D are described in [15].

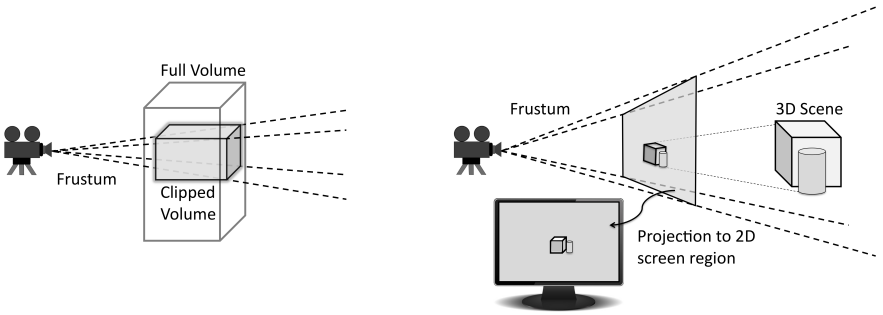
Uintah Software Framework: Uintah is a general purpose software framework used in the development of components for the numerical modeling of simulations involving fluid-structure interactions, computational fluid dynamics, solid mechanics and multi-physics simulation on core counts approaching 768K cores[19].

Uintah's native data output format, Uintah Data Archive (UDA), consists of individual timesteps (directories) each containing binary data and meta-data files written per MPI rank. The UDA format was originally designed in the mid 90s when large simulations were on the order of 2K cores. Several thousand files per directory were considered manageable. However, with core counts approaching 1 million cores, data output consisting of hundreds of thousands of small files is untenable from both a file system/OS and scalable I/O point of view. We have incorporated the PIDX library into the Uintah framework as a optional, high throughput, scalable I/O system for managing data output and checkpointing.

4 View-dependent Multiresolution Serial Reads

It is often desirable to utilize low-cost, low-power hardware to visualize massive simulation data as well as to perform cursory or user-directed analysis. For ex-

ample, a portable device could be used to monitor the progress of a simulation. Utilizing a hierarchical multiresolution data format enables view-dependent determination of the spatial extent and resolution of data to be loaded, permitting less data to be requested and facilitating interactive navigation of arbitrarily large datasets. In this section we will explain the mechanism of view-dependent data loading and the implementation of this technique in the VisIt visualization application. We compare read performance in a variety of scenarios.

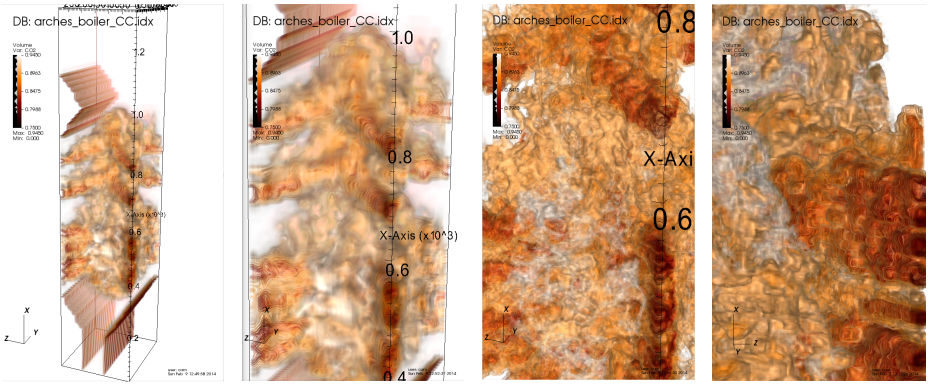


(a) (b)

Fig. 3: Example images showing the mechanism of view-dependent data reading. (a) Shows how the spatial extent of the data is clipped by the camera viewing frustum. Only data within the frustum must be loaded. (b) Shows the projection of a 3D scene onto the 2D camera viewing plane, which is copied to the display. The ratio of the area of a screen pixel to the projected area of a 3D volume unit is used to determine the resolution (HZ) level that will be loaded. For example, if 4 volume units project to one pixel, 1/4 resolution data is sufficient to convey the scene.

4.1 View-dependent data loading

All visualization systems incorporate some type of 2D or 3D camera model in order to transform data into a 2D image on a screen. The camera model can be distilled down to a matrix multiplication applied to the individual components of the data in order for them to appear in the desired location when projected onto the 2D viewing plane. A comprehensive description of the viewing pipeline is outside the scope of this work, but the process is explained in most introductory computer graphics texts such as [23, 12, 10]. The matrix resulting from the composition of model, viewing and projection transformations, as well as the near and far clipping planes, creates a viewing "frustum", a parallelepiped oriented in the world space coordinate system. The frustum is used to crop data outside the view of the virtual camera (called "clipping") and the ratio of the



(a) (b) (c) (d)

Fig. 4: Images showing view-dependent data loading for a Uintah combustion simulation. From left to right the view is continually refreshed while the camera is zoomed closer to the data.

projected size of a data unit compared to a screen pixel can be used to determine the optimal resolution of data to be loaded.

View-dependent data loading begins by applying the frustum clip planes to the volume of data being visualized. Clipping the bounds of the data volume with the viewing frustum is how we determine the spatial extent of the data to be loaded. See figure 3(a). Next we use the relative size of a display pixel compared to the size of a unit of the data volume projected to the screen in order to determine the minimum resolution necessary to visualize the data without artifacts. See figure 3(b). This technique is similar to the ubiquitous MIP-mapping technique of [27] used to decide what level of a given texture should be sampled based on its projection to screen space. Using the extent of the volume and relying on the fact that every HZ level reduces the resolution by half in one direction, we can determine the minimum level necessary for a unit of the data volume to cover approximately one pixel on the screen. Loading higher resolution data than this is a waste of time and memory because multiple data units are simply averaged over the same pixel, and loading a lower resolution will result in an overly coarse or "jaggy" image.

The combination of these two techniques allows for interactive exploration of any size data using even modest hardware resources. We describe the mechanism by which we have incorporated view-dependent rendering into the VisIt visualization system (see figure 4).

4.2 Implementation in VisIt

We have incorporated the IDX format as both a serial and parallel database reader plugin for VisIt.

Loading an entire large data volume is frequently excessive for simple visualization or cursory analysis and requires significant compute resources. For summary analysis and visualization, or to load extremely large datasets using more modest hardware, a coarse approximation of the data can be sufficient. Our implementation of the serial IDX reader in VisIt facilitates loading coarse levels of the data by selecting a sub-volume and resolution sufficient to accommodate a given viewing frustum. Additionally, utilizing the IDX format with VisIt enables a new capability for remote visualization: loading remotely hosted volumes is now possible by simply specifying the URL to be loaded. The remote server that provides IDX data has no knowledge of VisIt nor is there a need for VisIt to be installed at the remote location.

4.3 Performance Evaluation

Interactivity is the overall performance goal of using view-dependent data loading. The key idea is to load as little data as possible at one time while still providing sufficient coverage of the viewing region. Efficient data loading is an important component of engineering an interactive application, but additional support is required at every level. For example, it is important to frequently check for user input even during a complicated analysis or visualization operation or while reading data.

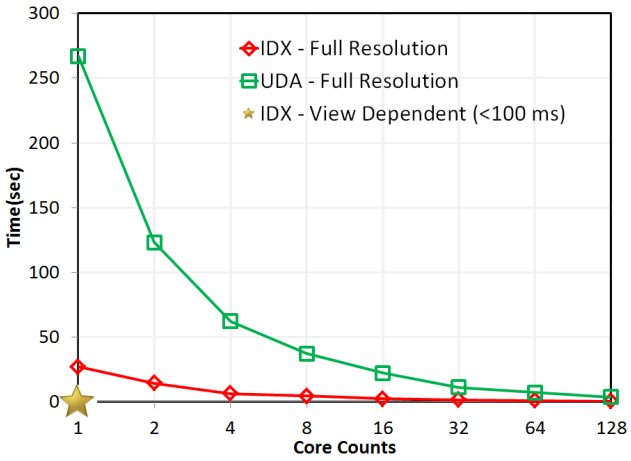


Fig. 5: VisIt load times on Tukey for a Uintah Data Archive (UDA) file versus the same simulation stored in the IDX format for increasing numbers of cores. The star in the bottom left indicates the time for loading the data using the serial view-dependent method.

Figure 5 shows the results of our experiments on Tukey. The times required by VisIt for loading a Uintah Data Archive (UDA) file versus the same simulation stored in the IDX format, are plotted using increasing numbers of cores. In order to compare view-dependent loading of multiresolution data to the existing practice of loading full resolution data using a visualization cluster, the point noted with a star at the bottom of the figure shows the average time for loading

the data using serial view-dependent visualization. All load times were gathered using the '-timings' argument to VisIt. The dataset shown was computed using 9920 processors and consists of 19 variables on a 1323×335×290 domain over 171 timesteps totaling approximately 4 TiB. The data shown in the figure is for loading a single variable of the dataset. Note that the differences observed in loading UDA vs IDX data at full resolution are due to the nature of the two file formats. IDX data is stored in a cache-oblivious layout, whereas data in UDA data is stored in naïve row-major order using one file per process. Because the UDA format uses so many files, additional access time (open/close/seek) is incurred compared to the IDX format, which uses fewer files to store the same data. UDA is comparable to one file per process I/O, so as more cores are used the overhead incurred in accessing files is reduced because the number of files that are accessed per process decreases. Most importantly, as can be seen in the chart, view-dependent data reading is significantly faster than loading full resolution data using any number of cores.

5 Parallel Reads at Full Resolution

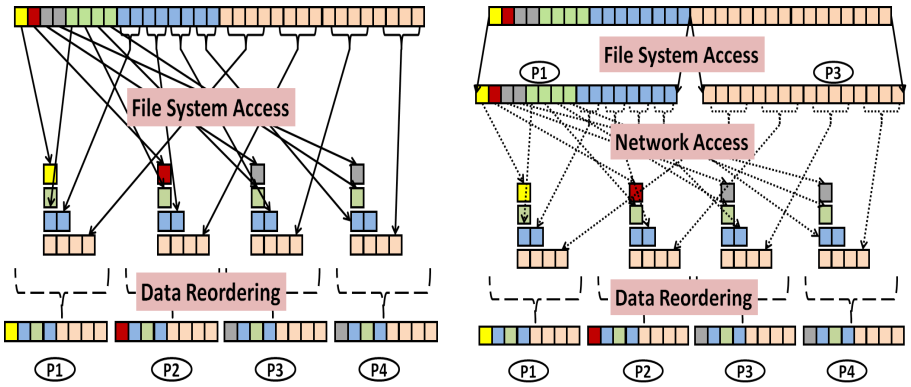


Fig. 6: Reading IDX: data is read into a hierarchy of z-buffers and then reordered to the conventional multidimensional application layout. (a) The one-phase approach that requires many small non-contiguous disk accesses to load data into the z-buffer hierarchy of each process. (b) The improved two-phase method. First, large contiguous blocks of data are read from disk by a few aggregators. Next, the data is loaded via the network into the z-buffer hierarchies of each process.

Until now we have demonstrated the utility of multiresolution reads in serial. However, full-resolution parallel reads are still important for simulation restarts

and comprehensive postprocessing analysis. In this section we describe the read implemenation of the PIDX library followed by its in-depth evaluation showing both strong and weak scaling results. We also compare PIDX performance with two common distributed parallel file formats, MPI collective I/O [24] and one file per process S3D I/O.

5.1 Read Implementation

Parallel IDX reads involve translation from the hierarchical progressive data order of the IDX format to the conventional multidimensional data layout suitable for use by application processes. We use our experience with the PIDX writer to design the corresponding reader [13]. We first describe a naïve implementation that uses one I/O phase to fetch data. Next, we improve upon this strategy by adding an additional data aggregation phase to achieve scalable performance by reducing noncontiguous data access.

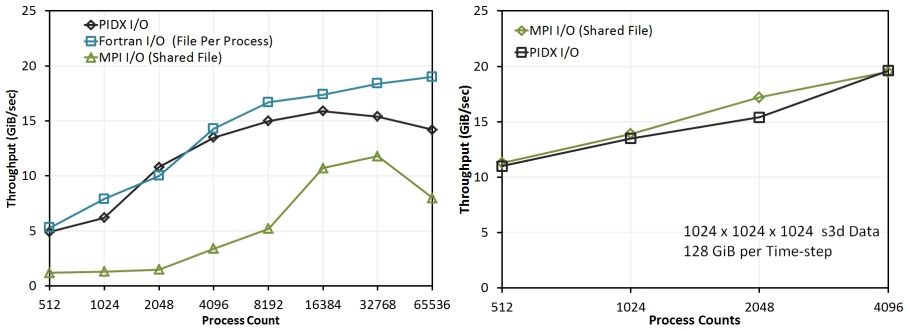
For the one-phase parallel reader, every process first reads the data at each HZ level (recall figure 1 from section 2.1), and then correctly orders the data for application usage. This strategy is complementary to the approach followed in one-phase writes, where every process first calculates the HZ order and corresponding HZ level for its sub-volume, and then uses independent MPI-I/O write operations to store each level. For both reading and writing this method entails a high degree of noncontiguous data access of the file system, dramatically impeding scalability [15]. See figure 6a.

To improve upon the naïve one-phase approach, we devised a two-phase I/O algorithm to mitigate the problems caused by large numbers of small-sized disk accesses. With this approach, only a select few processes assigned as aggregators access the file system, making large-sized contiguous read requests. Once the data is read by the aggregators, it is next transferred to every process over the network for re-ordering. As with PIDX writes, one-sided MPI communication is used for this purpose: after the aggregators finish reading data from the file system, all the processes use MPI_get() to gather data directly from the aggregators' remote buffers. Figure 6b illustrates the two-phase I/O approach.

5.2 Performance Evaluation

We perform two sets of experiments: weak scaling and strong scaling. Weak scaling increases the data size proportional to the number of processors. It is a useful measurement for determining the efficiency of reading data for simulation restarts that typically require reading a complete dataset using many cores. Strong scaling increases the number of processors while maintaining the same problem size. It is a useful metric to determine how much a particular task can be accelerated by the addition of computational resources. Strong scaling is an indicator of parallel visualization and analysis performance because these tasks are often performed using fewer cores than the original simulation.

Weak Scaling (simulation restarts): We use the S3D I/O simulator to benchmark the performance of parallel reads. We compared PIDX performance



(a) (b)

Fig. 7: [S3D I/O Benchmark] results using Edison (a) weak scaling of different I/O mechanisms including PIDX, Fortran I/O, and MPI I/O reading datasets with a block size of 32^3. (b) Strong scaling for PIDX and MPI I/O for reading datasets of dimension 1024^3.

with that of both the Fortran I/O and MPI collective I/O modules in S3D. With Fortran I/O, data is present in its native format organized into files equal to the number of processes used to create them. PIDX determines the number of files based on the size of the dataset rather than on the number of processes [15]. In the case of MPI I/O, data exists in a single, shared file. As opposed to Fortran I/O, both PIDX I/O and MPI I/O have an extensive data communication layer as part of the collective I/O phase. Default file system striping parameters were used for Fortran I/O, whereas for MPI I/O and PIDX I/O the Lustre striping was increased to span all 96 OSTs available on that system.

In order to benchmark read performance we first used the S3D simulator to create datasets of the corresponding format at each process count. We then invoked the S3D postprocessing module to enable parallel reads and conducted our experiments. For each run, S3D reads data corresponding to 25 timesteps. Every process reads a 32^3 block of double-precision data (4 MiB) consisting of four variables: pressure, temperature, velocity (3 samples) and species (11 samples). We varied the number of processes from 512 to $65,536$, thus varying the amount of data read per timestep from 4 MiB to 256 GiB.

Considering the performance results in figure 7a, we observe that PIDX scales well up to $32,768$ processes, and that for all process counts, it performs better than MPI I/O. We also observe that both PIDX and MPI I/O do not perform as well compared to Fortran I/O because unlike PIDX and MPI I/O, Fortran I/O does not require any data exchanges across the network. The performance pattern of PIDX is mainly due to its aggregation phase, which finds a middle ground between the single-shared-file MPI I/O and one file per process I/O.

One new observation that we make for parallel I/O is the decline in scalability for both MPI I/O and PIDX at higher core counts, whereas Fortran I/O

shows continued scaling. This decline is in contrast to parallel writes for which continued positive scaling results have been demonstrated for all I/O formats [14]. One possible explanation for this behavior is that network communication involved in the collective I/O data scattering phase may play a role in this effect. If this were the case, we note that Fortran I/O would not be affected since each process is independent of the network. We believe the poor scaling of data reads will become increasingly problematic in the future and that further investigation may be worthwhile.

Strong Scaling (Post-processing analysis and visualization): Using the S3D I/O simulator, we compared the strong scaling results of PIDX with MPI I/O for 25 timesteps on Edison. We used the S3D simulator to create input datasets of the corresponding format using 32768 cores with a per process domain size of 32^3. We then invoked the S3D postprocessing module to enable parallel reads and conducted our experiments. For each run, S3D reads data corresponding to 25 timesteps. Because the S3D simulator does not make available the capability to read multiple files per process, we were unable to acquire strong scaling numbers for Fortran I/O. Our experiments used a 3D domain of dimension 1024^3. We varied the number of processes from 512 to 4096, for which the block size per process ranged from 128^3 (512 processes) down to 64^3 (4096 processes). The results of this study are shown in Figure 7b. We observe that PIDX and MPI I/O perform similarly at all core counts.

6 Parallel Reads at Varying Resolution

For our final experiments we explore parallel reading of multiresolution data. We already discussed serial multiresolution reads in Section 4, and parallel full resolution reads in Section 5. Providing parallel multiresolution data loading confers the advantages of faster coarse resolution data access while allowing for greater processing power and total aggregate memory available on larger clusters.

Parallel multiresolution reads can be useful for visualization and postprocessing analysis. As with serial multiresolution data reading, relatively small compute clusters can be utilized to perform analysis or to visualize much larger datasets than was previously possible. In addition, dedicated visualization clusters such as Tukey may have associated GPU arrays or be connected to large displays such as those used for power walls or immersive caves. These clusters may be utilized to produce ultra-high-resolution imagery using rendering techniques such as raytracing or to perform postprocessing analysis using a mixture of GPU and CPU resources.

6.1 Design

One of the major challenges for performing multiresolution reads in parallel is ensuring stable scalable performance by limiting data reads. We made two significant changes to extend parallel full resolution reads to support multiresolution capabilities. First, based on the desired level of resolution we added the capability to directly select the appropriate HZ levels. Second, we made the aggregation

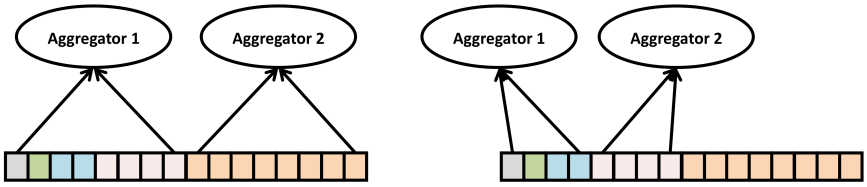


Fig. 8: Examples showing aggregation balancing for parallel multiresolution reads: (a) aggregator assignment when reading full-resolution data, and (b) aggregator assignment when reading half resolution data.

phase self-balancing by maintaining the number of aggregators for varying levels of resolution. See figure 8. By balancing the load across aggregators, we ensure a uniform reduction of workload.

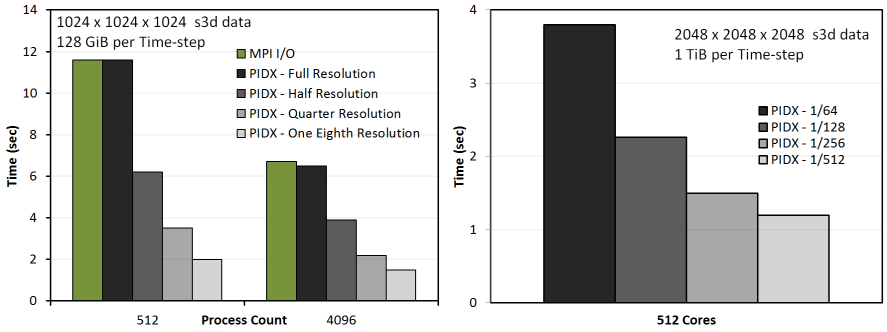


Fig. 9: [S3D I/O Benchmark] results using Edison. (a) Timings for reading full resolution 1024^3 volume data using PIDX and MPI I/O as well as timing for partial resolution reads using PIDX. (b) Timings for reading a 2048^3 volume data at partial resolutions using PIDX, simulating an environment in which it is not possible or desirable to read the full resolution data.

6.2 Performance Evaluation

Due to limitations in aggregate memory, loading data at full resolution would not be feasible for some datasets, but with multiresolution support, data can be loaded at coarser resolutions, enabling approximate analysis or visualization for extremely large datasets.

We conducted these experiments using Edison, with stripe setting similar to our parallel full resolution experiments. We first used the S3D simulator to create input datasets of global resolution 1024^3 and 2048^3. For the 1024^3 volume we evaluate the performance of multiresolution reads by first reading the data at full resolution followed by reading the data at partial resolution. We decrease

the resolution by half for each successive test down to 1/8 of the original volume size. For comparison, we also show the corresponding result for loading the full resolution volume using MPI I/O (see figure 9(a)). In accordance with our strong scaling results, we observe that reading the full resolution data requires approximately the same amount of time for both PIDX and MPI I/O. For multiresolution reads we see almost perfect scaling when the resolution is reduced by half, but as the resolution continues to decrease the efficiency begins to deteriorate due to the reduced workload of individual aggregators as the requested data size decreases.

We use the 1 TiB per timestep 2048^3 volume to simulate the situation in which the amount of data to load might be larger than the aggregate system memory. For this particular case we begin reading data at 1/64 resolution and decrease by half for each run down to 1/512. The results for this experiment can be see in figure 9(b). From the figure we observe that the read time using 512 cores is less than four seconds, which is fast enough to be used for cursory visualization or to compute approximate analysis on an otherwise unweildy dataset. Also, as observed with the 1024^3 case, we notice a nearly perfect scaling for the first decrease in resolution while successive reductions are less optimal.

7 Conclusion

We identified two broad classes of data reading: full resolution reads for simulation restarts or postprocessing analysis and partial resolution reads of spatial subsets suitable for cursory analysis and visualization. We have shown that using a hierarchical multiresolution data format enables scalable and efficient serial and parallel read access. The technique of view-dependent reading was incorporated into the VisIt visualization application to enable interactive visualization and analysis of massive datasets with very modest hardware resources. Two different application codes, S3D and Uintah, were used to compare native data formats with the IDX hierarchical multiresolution data format. We demonstrated that reading using PIDX weak scales well to 32K cores and is approximately 25% slower than Fortran I/O and approximately 50% faster than MPI I/O. We noted a falloff in scaling performance of reads for both PIDX and MPI I/O beyond 32K cores. We also demonstrated the strong scaling characteristic of PIDX. Finally, we described the use of parallel multiresolution reads to support faster visualization and approximation of comprehensive analysis when dealing with very large datasets.

8 Acknowledgements

This research used resources of the National Energy Research Scientific Computing Center and the Argonne Leadership Computing Facility at Argonne National Laboratory, supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357 and DE-AC02-05CH11231, respectively. It is based upon work supported by the Department of Energy, National Nuclear Security Administration, the Department of Energy Scalable Data Management, Analysis, and Visualization (SDAV) SciDAC Institute and PISTON, award numbers DE-NA0002375, DE-SC0007446, and DE-SC0010498, respectively.

References

1. HDF5 home page. http://www.hdfgroup.org/HDF5/.
2. OpenGL standard. http://www.opengl.org/.
3. OpenGL view frustum culling. http://www.lighthouse3d.com/tutorials/view-frustum-culling/.
4. VAPOR home page. http://www.vapor.ucar.edu/.
5. Visit home page. https://wci.llnl.gov/codes/visit/.
6. J. P. Ahrens, J. Woodring, D. E. DeMarle, J. Patchett, and M. Maltrud. Interactive remote large-scale data visualization via prioritized multi-resolution streaming. In *Proceedings of the 2009 Workshop on Ultrascale Visualization*, UltraVis '09, pages 1–10, New York, NY, USA, 2009. ACM.
7. J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. M. Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using s3d. In *Computational Science and Discovery Volume 2*, January 2009.
8. T.-c. Chiueh and R. H. Katz. Multi-resolution video representation for parallel disk arrays. In *Proceedings of the first ACM international conference on Multimedia*, MULTIMEDIA '93, pages 401–409, New York, NY, USA, 1993. ACM.
9. J. M. del Rosario, R. Bordawekar, and A. Choudhary. Improved parallel I/O via a two-phase run-time access strategy. *SIGARCH Comput. Archit. News*, 21:31–38, December 1993.
10. J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice (2Nd Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.
11. S. Guthe, M. Wand, J. Gonser, and W. Strasser. Interactive rendering of large volume data sets. In *Proceedings of the conference on Visualization '02*, VIS '02, pages 53–60, Washington, DC, USA, 2002. IEEE Computer Society.
12. D. Hearn and M. P. Baker. *Computer graphics, C version*, volume 2. Prentice Hall Upper Saddle River, 1997.
13. S. Kumar, V. Pascucci, V. Vishwanath, P. Carns, M. Hereld, R. Latham, T. Peterka, M. Papka, and R. Ross. Towards parallel access of multi-dimensional, multi-resolution scientific data. In *Petascale Data Storage Workshop (PDSW), 2010 5th*, pages 1–5, 2010.
14. S. Kumar, V. Vishwanath, P. Carns, J. A. Levine, R. Latham, G. Scorzelli, H. Kolla, R. Grout, R. Ross, M. E. 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*, SC '12, pages 50:1–50:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.
15. 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 the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, pages 103–111, Washington, DC, USA, 2011. IEEE Computer Society.
16. J. K. Lawder and P. J. H. King. Using space-filling curves for multi-dimensional indexing. *Lecture Notes in Computer Science*, 1832:20, 2000.
17. J. Li, W.-K. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. Parallel netCDF: A high-performance scientific I/O interface. In *Proceedings of SC2003: High Performance Networking and Computing*, Phoenix, AZ, November 2003. IEEE Computer Society Press.

18. J. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments, CLADE '08*, pages 15–24, New York, June 2008. ACM.
19. Q. Meng, A. Humphrey, J. Schmidt, and M. Berzins. Investigating applications portability with the uintah dag-based runtime system on petascale supercomputers. In *Proceedings of the 2013 ACM/IEEE conference on Supercomputing (SC '13)*. ACM, 2013.
20. V. Pascucci and R. J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2001.
21. 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.
22. V. Pascucci, G. Scorzelli, B. Summa, P.-T. Bremer, A. Gyulassy, C. Christensen, S. Philip, and S. Kumar. The visus visualization framework. In E. W. Bethel, H. Childs, and C. Hansen, editors, *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*, chapter 19, pages 401–414. Chapman & Hall and CRC Computational Science, 2012.
23. P. Shirley and S. Marschner. *Fundamentals of Computer Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 3rd edition, 2009.
24. R. Thakur, W. Gropp, and E. Lusk. On implementing MPI-IO portably and with high performance. In *In Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems*, pages 23–32. ACM Press, 1999.
25. 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. IEEE, 2013.
26. C. Wang, J. Gao, L. Li, and H.-W. Shen. A multiresolution volume rendering framework for large-scale time-varying data visualization. In *Fourth International Workshop on Volume Graphics, 2005*, pages 11 – 223, June 2005.
27. L. Williams. Pyramidal parametrics. *SIGGRAPH Comput. Graph.*, 17(3):1–11, July 1983.