Extreme Event Analysis in Next Generation Simulation Architectures

Stephen Hamilton[1], Randal Burns[1], Charles Meneveau[2], Perry Johnson[2], Peter Lindstrom[3], John Patchett[4], and Alexander S. Szalay[5]

[1] Department of Computer Science
Johns Hopkins University
Baltimore, Maryland 21218
{stephenh, randal}@cs.jhu.edu
[2] Department of Mechanical Engineering
Johns Hopkins University
Baltimore, Maryland 21218
{meneveau, pjohns86}@jhu.edu
[3] Lawrence Livermore National Laboratory
Livermore, CA 94550
pl@llnl.gov
[4] Los Alamos National Laboratory
Los Alamos, NM 87545
patchett@lanl.gov
[5] Department of Physics and Astronomy
Johns Hopkins University
szalay@jhu.edu

Abstract. Numerical simulations present challenges because they generate petabyte-scale data that must be extracted and reduced during the simulation. We demonstrate a seamless integration of feature extraction for a simulation of turbulent fluid dynamics. The simulation produces on the order of 6 terabytes per timestep. In order to analyze and store this data, we extract velocity data from a dilated volume of the strong vortical regions and also store a lossy compressed representation of the data. Both reduce data by one or more orders of magnitude. We extract data from user checkpoints in transit while they reside on temporary burst buffer SSD stores. In this way, analysis and compression algorithms are designed to meet specific time constraints so they do not interfere with simulation computations. Our results demonstrate that we can perform feature extraction on a world-class direct numerical simulation of turbulence while it is running and gather meaningful scientific data for archival and post analysis.

1 Introduction

Supercomputing trends toward exascale motivate our research, specifically the increasing performance gap between processing and I/O. At exascale, simulations will output fewer than one byte for every 10^5 bytes of system state; they

will produce 200-300 PB/s in memory [1] and only 1 TB/s [2] will be saved to persistent storage. Next generation architectures must define meaningful ways to output data that preserve scientific discovery on reduced data representations. The Trinity supercomputer at Los Alamos has deployed a burst buffer architecture [3] to fill the performance gap between cluster memory and disk filesystems. Burst buffers place the SSD storage on the fast network to catch I/O bursts that would overwhelm the filesystem. Data on burst buffers are short lived; they must be discarded or stored to file system in (tens of) minutes. Our experiments run on a research cluster with nodes designed to mimic the performance of Trinity's burst buffers and reduce data by orders of magnitude while preserving usable data for visualization and extreme event analysis [4]. Our experiences using the Johns Hopkins Turbulence Databases [5] (JHTDB) inform the choice of data products that we extract from burst buffers. Specifically, we create compressed lower precision representations of the full field velocity data and extract high-resolution velocity data from regions of relatively high vorticity. JHTDB contains multiple datasets from direct numerical simulations that range from tens to 150 terabytes. In particular, the isotropic turbulence dataset contains 5028 timesteps of velocity with three components of floating point values and one component of floating point pressure values on a 1024^3 spatially dense regular grid. This dataset provides scientists all over the world an opportunity to discover many aspects of turbulence without the need to run their own large simulation. A number of discoveries from the JHTDB have come from the combination of visualization and analysis of high vorticity regions. These include a vorticity hierarchy that is not evident on smaller scale simulations [6] and that magnetic flux freezing in high-conductivity plasmas fails in the presence of MHD turbulence, explaining why solar flares can erupt in minutes or hours rather than the millions of years predicted by flux freezing [7].

Going forward, the lack of I/O bandwidth to long term storage will slow down the simulation. Transferring output every few timesteps from a larger simulation (8192^3) would slow down the simulation by an order of magnitude; JHTDB's isotropic database stores every tenth timestep because the integration time-step for stably solving the system is smaller than that needed for analyses. Each timestep serves as a checkpoint which is utilized if a restart is required due to simulation failure. The I/O needed to checkpoint simulations to file systems has become the performance limiting workload in scalable HPC [8] and exposure to failure governs checkpoint frequency; they are taken much less frequently than needed for time-resolved analysis of the simulated processes of turbulence.

We develop methods that capture and extract relevant scientific data of a direct numerical simulation as it runs. We propose a model in which checkpoints are written to burst buffers at the frequency needed for analysis and then we extract a subset of the data and reduced representations that can be utilized for scientific analysis in real-time as well as post simulation. The extraction requires little processing power and it does not disrupt the running simulation. On Trinity [9], the burst buffers are located on additional nodes that are separate

from compute nodes. Burst buffers in recent architectures collocate compute and SSDs [10] and extraction codes can be run within the burst buffer nodes.

In the first part of the process, we extract a subset of velocity data in 3D space only at points where the vorticity magnitude exceeds a defined threshold.[6] Next we dilate the volumes within this 3D space by a kernel size based on the requirements for post analysis and extract the velocity field in the dilated regions. The dilation allows us to capture data just outside the high vorticity regions needed for iso-surface extraction and Lagrangian interpolation in post-processing. Many filters and derivative equations also rely on this additional data gained from the dilation for interpolation kernels around the region, which makes the extracted data useful for scientific analysis.

This method deliberately leaves out regions of low vorticity. Understanding that we cannot save the entire dataset, we extract a separate dataset that contains full field lower precision data by using lossy compression. We leverage the zfp algorithm [12], which is specifically designed to compress floating point scientific data in 1D, 2D, or 3D space. zfp's lossy compression is *error-bounded*; it guarantees that the values differ from the original by less than a specified amount. zfp achieves an order of magnitude or more compression and the loss of accuracy is indistinguishable when visualizing the data. These characteristics lend themselves well to capturing exascale simulation data for visualization.

Combining these extraction techniques allows one to visualize the simulation while it runs and create an archival database that is exact in regions of high-vorticity and error-bounded elsewhere. The data products are an order of magnitude or more smaller than simulation output and suitable for scientific post analysis. Although our focus is on vorticity and velocity data from direct numerical simulation of the single-phase incompressible Navier Stokes equations, the velocity extraction technique generalizes to richer fluid mechanical simulations that may include magnetic field, magnetic potential and density, to other governing equations, such as Large Eddy Simulations, and to numerical simulations from other domains that run on regular and irregular grids, such as climate, material fracture, and combustion.

Our evaluation utilized 32 burst buffer nodes that contained either 4, 10, or 16 cores. For extracting velocity in high vorticity regions, we reduce an 8192^3 grid by one order of magnitude in under 10 minutes (the 4 core node was not able to meet this time constraint). For lossy compression of velocity for the entire grid, we reduced by one order of magnitude and it took approximately 10 minutes for a single timestep. These results inform us that having 32 burst buffer nodes with a minimum of 10 cores each would allow us to execute either extraction task in under ten minutes for a world-class turbulence Direct Numerical Simulation (DNS).

[6] Thresholds are easy to choose because turbulence has threshold values with physical meaning derived from the inverse Kolmogorov scale that describes the near absence of, medium, and high vorticity [11]

2 Related Work

Supercomputing continues to evolve with speed increases and hardware architecture changes that coincide with application development to leverage these new architectures. Bent et al. [13] explore burst buffer configurations and demonstrate that placing SSDs between compute nodes and the storage array allow jitter-free co-processing of their visualization tasks and reduce total time to completion by up to thirty percent. We utilize a similar architecture in our work. Ma et al. [14] discuss in-situ data extraction and visualization. They modify the simulation code to provide data useful for visualization in-situ, whereas our work performs feature extraction in-transit via burst buffers without having to modify existing simulation codes. Ahrens et al. [15] describe and test methods of utilizing multi-core CPU and GPU based processors in the Roadrunner supercomputer to perform visualization of an exascale simulation in-situ. Chen et al. [16] utilize the HemeLB lattice-Boltzmann code for large-scale fluid flow. They discuss pre- and post-processing along with computational steering to modify simulation parameters in situ. This work differs from ours in the way the data is saved and utilized for post-processing. They create a multi-resolution data structure by storing their simulation output in a hierarchical order. This method allows for visualization without reading the entire dataset. In our work, we utilize the SSD burst buffers to read the entire timestep and perform thresholding and extraction of high-magnitude events on a per-timestep basis. Wang et al. [17] developed a file system (BurstFS) that aggregates I/O bandwidth from burst buffers and maintains a distributed key-value store of metadata for the files. This system allows an application to perform small non-contiguous read operations on the burst buffer. Because our feature extraction reads of all the data, this file system would not benefit our work.

We build upon the concept of burst buffers [18] to integrate non-volatile memory into the supercomputing storage hierarchy. We focus specifically on using the SSD to capture write bursts, particularly those from checkpoint workloads. Other concept papers have discussed using burst buffers more generally in the HPC memory hierarchy [19].

3 Problem Overview

Extreme scale simulations produce petabyte-scale data that must be read for feature extraction and/or down-sampling in real time without hindering the computation of the underlying simulation. This presents a host of challenges due to the competition for memory resources, bandwidth, I/O, and storage. The burst buffer architecture adds to the storage hierarchy to enable a temporary storage area in between permanent storage and resident RAM for fast reading and writing. Once a timestep is written, a feature extraction application must read, process, and store the extracted data prior to the simulation overwriting the burst buffer space with a subsequent timestep. Typically, burst buffer capacity is chosen to be more than two times cluster memory so that the burst buffer

will hold two to three consecutive timesteps. In addition, the data reduction and the simulation must maintain synchrony throughout the simulation. The feature extraction must process data prior to the simulation writing the next timestep.

The primary problem with world-class numerical simulation data is the sheer size of the output. Our target fluid simulation has a desired output (not every solved timestep) with dimension 8192^3 over 4000 timesteps and produces 4 attributes: 1 component for pressure and 3 for a velocity vector with 4 byte floating point (single precision) values resulting in total data of

$$8192^3 * 4 * 4 * 4000 = 3.518 \times 10^{16}$$

bytes, or approximately 31.25 petabytes of data. Although our simulation generates a pressure field, the remainder of this paper focuses only on storing the velocity data. Each timestep contains about 6 terabytes of velocity data that must be read, processed, and written on the order of tens of minutes. This process must complete in time in order to maintain synchronization with the simulation to ensure data extraction successfully finishes. Any delay would cause the simulation to stall.

The data extraction itself presents a complex problem, because it takes computational power to perform the extraction and it must complete in a timely fashion. If we were to do in-situ analysis, extraction resources compete directly with the processing required to perform the simulation. Our method leverages burst buffers in order to perform this extraction in-transit without interfering with the simulation.

4 Methods of Extraction

We present our two methods of extraction: velocity data in regions of high vorticity and lossy compression of the full field. Extraction produces two datasets, each an order of magnitude or more smaller than simulation output. Prior to presenting methods, we motivate our use of Q-criterion for identifying vortices.

4.1 Calculating Highly-Vortical Regions

In turbulent flows, identification of coherent structures, specifically vortices, aids in scientific understanding of these flows. Inside and around these high vortical regions, energy dissipation and squared vorticity (enstrophy) are orders of magnitude higher than the mean values, which we refer to as extreme events [20]. There are various methods for identifying vortices. Vortices are defined by the velocity field that reflects the rotational qualities and there is not a single approved method to describe vortices. Dubief and Delcayre [21] examine four methods of vortex identification: pressure, vorticity magnitude, λ_2, and Q-criterion. Because pressure fails to capture fine details in isotropic turbulence [21] and λ_2 appears to be affected by small noise present in all data, we examine visualizations based on vorticity magnitude and Q-criterion. Each of these two

methods provide good visualizations of vortical flow structure when utilized to generate iso-surfaces. However, one particular issue with vorticity magnitude is that the vorticity criterion does not distinguish between swirling motions and shearing motions. Thus, vorticity magnitude can also present layered structures that are vorticity sheets and not vortices [22]. Q-criterion is also not perfect as it fails to reliably identify Bödewadt vorticies. However note that such vorticies occur normal to a wall, and our isotropic turbulence dataset is periodic and does not contain any walls [23]. We chose to compare the performance of the vorticity magnitude and Q-criterion for generating vortical flow iso-surfaces on an isotropic turbulence dataset.

In order to compare Q-criterion versus vorticity performance we defined a threshold that is equivalent for each calculation. The thresholds and resulting data can be constrained based on either scientific concerns (the loss of accuracy when evaluating averages of gradient norms over the entire flow volume) or system resources that set a target data size. This adjustment allows us to produce data that fits within available storage in the computing center, while still gathering useful scientific data to study these high vorticity regions. In order to determine the threshold, we begin by using a multiple of the root-mean-square value of the vorticity fluctuations. This value is known a-priori, based on knowledge of the dissipation rate ϵ and fluid viscosity ν according to $\langle \boldsymbol{\omega} \cdot \boldsymbol{\omega} \rangle^{1/2} = \sqrt{\epsilon/\nu}$ [24] where ω is the vorticity vector (curl of the velocity). For the data from the JHTDB, this value is $\sqrt{.0928/.000185} = 22.4$, which is also the inverse Kolomogorov time scale τ_η. Since we are interested in high vorticity regions, we scale this low reference threshold to achieve a clear visual representation of high vorticity regions.

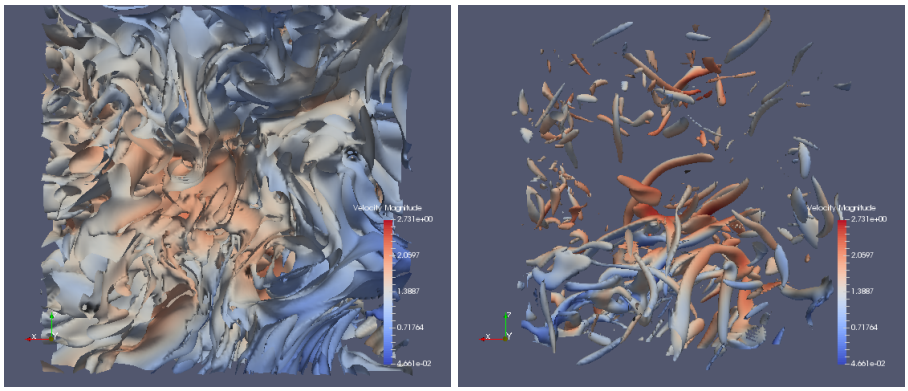

Fig. 1. Vorticity magnitude contour at threshold 22.4 (left) and 55.98 (right)

We tested various multiples of $1/\tau_\eta$ and found that a multiple of 2.5 presented clear vorticity structures without obvious erroneous surfaces. The threshold chosen in this case is $2.5 * 22.4 = 55.98$. The visualization of vorticity magnitude

at this threshold was a much clearer representation of vortices than using a threshold of 22.4 as seen in Figure 1.

Upon finding a reasonable threshold, we calculated the equivalent threshold for Q-criterion. In the absence of straining motions, the relationship between the threshold of vorticity and Q-criterion can be taken to be as follows: $Q = \frac{1}{4}\omega^2$. Therefore the threshold value for Q that we chose is $Q = .25(55.98)^2 = 783$.

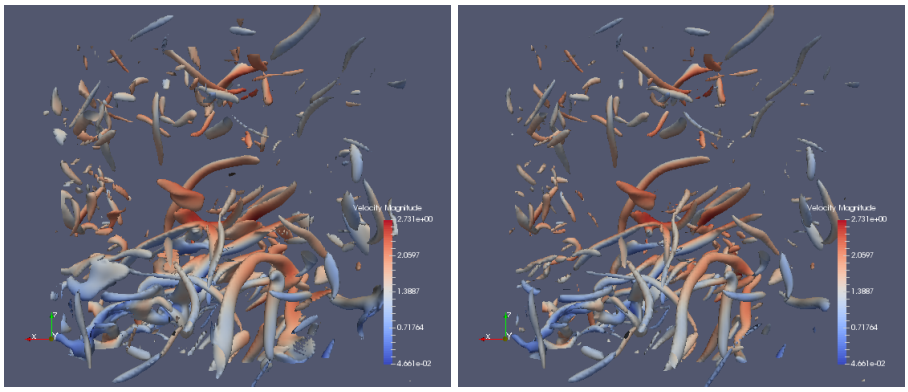

Fig. 2. Vorticity magnitude (left) and Q-criterion (right)

Figure 2 shows the visualization of vorticity magnitude contour versus the Q-criterion contour. Though they look very similar, the bottom left corner of the left image (vorticity magnitude) displays a structure that is not present in the Q-criterion visualization. This is due to shearing, because the vorticity magnitude does not differentiate between shearing and curl. In the definition of Q, strain is subtracted from vorticity which results in a lower Q value and filters out shearing. We performed additional tests at various thresholds and cube dimensions (subsets of the full 8192^3 grid) to determine whether the computation of Q-criterion or vorticity magnitude has an impact on overall feature extraction time. Table 1 compares total computation times, which includes reading from and writing to the burst buffer. Our results show that Q-criterion computes slightly faster than vorticity regardless of cube size. As a result of these

Cube Size	Vorticity Threshold	Q Threshold
64	.257	.222
128	1.597	1.375
192	5.070	4.500
256	11.120	9.522

Table 1. Vorticity vs. Q Thresholding in seconds total time per cube on a single core

considerations and tests, we choose to use Q-criterion for all analyses in the remainder of this paper. We also note that Q-criterion is generally accepted in the turbulence community for vortex identification.

4.2 Thresholded Vorticity Volumes

In order to capture the velocity data, we create a three-dimensional stencil that encompasses the regions of high-vorticity. This stencil masks out low Q regions and generates a sparse representation of velocity data within the regions. This sparse representation is a vtkUnstructuredGrid that consists of floating point coordinates in real space and the corresponding velocity vector at each point, thus each point contains six corresponding floating point values. This is not an ideal method of storage, however it works well enough in this application. Since the representation of vorticies in isotropic turbulence appear as worms, the goal is to capture velocities within all points within these worms, while discarding the velocity data outside of these structures. This data is the losslessly compressed to preserve the original values.

We begin by creating a stencil that "cuts out" high-vorticity regions from the full data. This includes points above the Q-criterion threshold. These regions are then dilated to include nearby points that are below the threshold. Dilating by four cells allows us to later compute most quantities of interest, including Q-criterion, vorticity magnitude, marching cubes for iso-surface extraction, velocity derivatives, and 4th-order Lagrangian interpolation. In order to create the stencil we create a bitmask dataset of the same dimensions of the original dataset and set all values above the threshold to one and those below to zero. Next we dilate this stencil with kernel size of four, meaning that each point that is already set to one sets all points within four voxels to one. Then we mask the velocity field with this zero/one data set, which extracts velocity values from the high vorticity regions and zeros out all other regions. The resultant data set contains a subset of velocity where each velocity vector retained contains a point coordinate to define its spatial location. The data can be utilized to reconstruct Q-criterion and iso-surfaces at or above the specified threshold. Figure 3 illustrates a visualization of dilated velocity volume utilizing the Q threshold of 783. Figure 4 demonstrates the ability to extract contours at higher thresholds from the thresholded velocity volume shown in Figure 3.

4.3 Lossy Compression

While thresholding works well for scientists studying events specifically within extreme vortical regions, it may be necessary to save information outside those regions for post analysis. For example, vortex precursors may occur in initially weak vortical regions, which then act as seeds for subsequent vortex intensification. In addition, a researcher may need information about conditions where velocity may be relatively high, which may not be contained in our thresholded data due to the fact that vorticity is a measurement of curl or rotation. We present a method for storing *all* of the data in a lossy compressed form for post

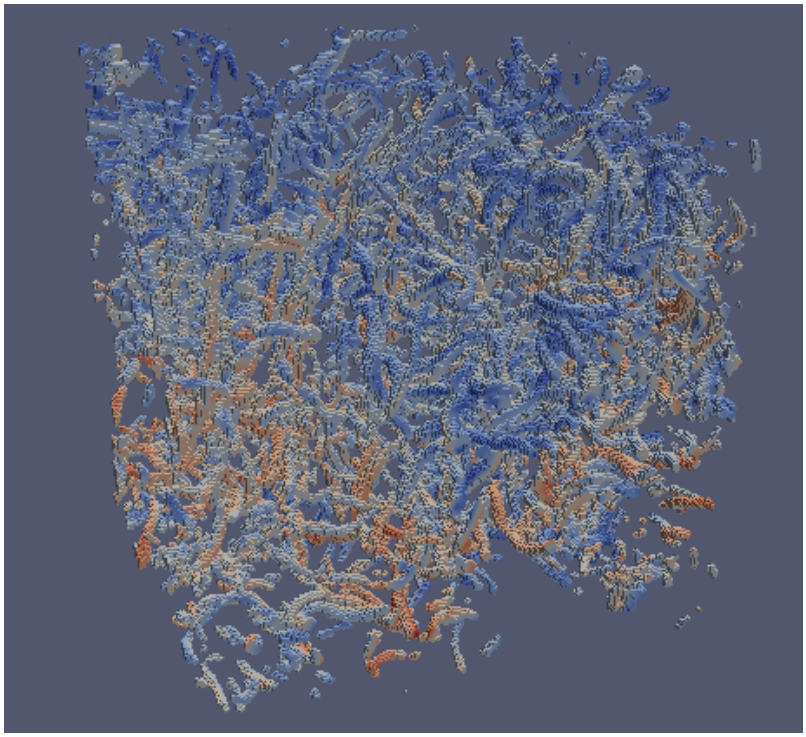

Fig. 3. Visualization of a 256 cube of dilated velocity in regions above Q threshold of 783

analysis and visualization. This does not provide exact raw simulation data, however, it provides data that is within a defined error tolerance. The error introduced on these data will be shown to be insignificant for the purposes of visualization, making the data desirable for post visualization analysis.

In order to store the data in a lossy form, we utilize a recent compression algorithm, zfp [12], designed specifically for the compression of multi-dimensional, floating-point scientific data. It contains various options for compression, one of which is to specify an absolute error tolerance. Utilizing this method at an error tolerance of 10^{-1} on the velocity data (the root-mean-square value of the velocity fluctuations is 0.686 while its mean is zero), we achieve an effective reduction of one order of magnitude from the raw velocity data. We note that this reduced dataset is intended for post analysis and visualization, and cannot be used as checkpoint data to restart the simulation.

While zfp operates on one scalar component, it contains a striding option that allows us to compress all three velocity vector components and store them as separate compressed blocks of data. We extended the Visualization Toolkit (VTK) [25] compression options to provide dimensions and component sizes to the compressor in order for zfp to have the information required to compress the

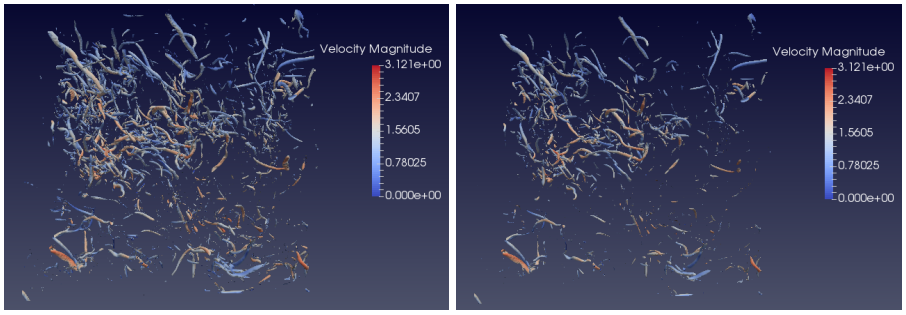

Fig. 4. Iso-surface extraction from dilated velocity threshold at Q thresholds 1700 (left) and 2500 (right)

data. Utilizing the striding option, each of the three velocity vector components (x, y, z) are compressed separately and stored as concatenated binary data. The data sizes for each axis are stored within the VTK XML file format as metadata in order for VTK to correctly decompress the data. During decompression, each component is decompressed into a separate array and interleaved back to their original representation creating a VTK float array of velocity vector values. If pressure or another scalar field were added, this could be compressed as well, and we would expect similar results.

5 Experimental Results

Experiments utilize the Visualization Toolkit version 7.1.0 by Kitware [25]. While we focus our experiments on finding high-Q vortical regions, VTK provides the flexibility of performing many other scientific computations on the simulation data. VTK provides a rich toolset for analysis and visualization. In addition to VTK, we utilize the zfp compression algorithm [12]. We compiled this natively into VTK in order to provide 3 dimensional lossy compression on VTK structured grid data.

We conducted all experiments on nodes in the Los Alamos National Lab development cluster called Darwin. We began by utilizing a partition built to emulate the performance of burst buffer nodes on Trinity which is used to test development software, such as the hierarchical input/output library [9]. Each node is equipped with a 6-core, 12-thread Intel Xeon E5-2630 2.30GHz processor, 128 GB of RAM, and an Intel P3700 400 GB SSD that is rated for 2.8 GB/s of sequential throughput and up to 460K random read IOPs.

5.1 Dilated Threshold

We perform a threshold and dilation velocity cutout operation on a cluster with SSD burst buffers that contains a single timestep of raw simulation data. We vary the cube size into which we decompose the problem in order to find the

cube size that maximizes throughput. Smaller cubes reduce I/O throughput and reduce skew and memory pressure. Larger cubes increase I/O throughput, but reduce the efficacy of caching, particularly on smaller processor caches up the memory hierarchy. We find that a cube size of 256^3 maximizes throughput for this computation (Table 2). Above 192^3, performance is stable and degrades slightly above 256^3, which we attribute to increased cache misses.

Size	Read	Q Comp	Thresh	Write	Total	Throughput
64	.029	.117	.0154	.0266	.222	13.51 MB/s
128	.043	.877	.064	.136	1.34	17.91 MB/s
192	.080	2.83	.206	.542	4.46	19.32 MB/s
256	.136	6.15	.399	1.08	9.522	20.17 MB/s
384	.373	20.68	2.23	5.67	34.337	18.87 MB/s
512	.788	48.76	5.31	13.17	78.86	19.47 MB/s

Table 2. Comparison of I/O and computation times in seconds when processing a single cube

Averaged over all cubes, the extracted thresholded velocity data is reduced by a factor of 29 times. The raw size of 256^3 of velocity data is 192 MB and the dilated extraction averages 6.7 MB. As mentioned previously, if we increase the threshold, the extracted data size will decrease.

Based on the throughput in Table 2, we measured the resources required to perform a dilated velocity extraction of an 8192^3 grid on the order of ten minutes. Our results show that this can be done utilizing 32 nodes with 10 cores each (320 cores) and 6 TB of SSD storage to achieve a full extraction of the data within the time constraint.

Next, we performed read tests on the 32 heterogeneous nodes to compare the local hard drive throughput with the SSD. Of these nodes, 15 did not contain a spinning hard drive therefore there are no hard disk throughput results for those nodes. This test demonstrates the performance gains of the burst buffer by comparing it to the node's local hard disk. The burst buffer was between 5 to 20 times faster depending on the node as shown in Figure 5.

5.2 Lossy Compression with zfp

The compression algorithm zfp provides an order of magnitude reduction by compressing scientific floating point data where the values spatially near each other have low variance. Although it provides many features like in-memory compression, we specifically use it for compressing data for storage with a predefined lossy tolerance of 10^{-1}. At this tolerance we achieve an order of magnitude of compression with visually lossless reconstruction, which is far superior to the default ZLib library utilized in VTK. We also note that while the error threshold is set at 10^{-1} our reconstructed data maximum error was .017. Table 3 shows

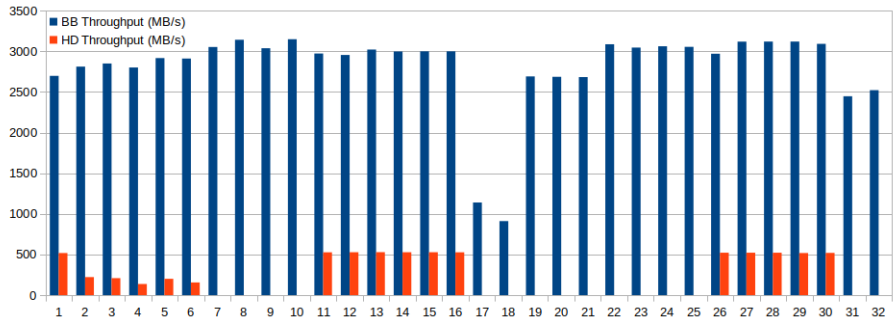

Fig. 5. Burst buffer throughput compared to hard disk throughput

the resulting size of compressing different sized cubes of isotropic turbulence data, along with the amount of time required to compress the cube. Visually the results of the lossy compression are indistinguishable from the original data as show in Figure 6.

Cube Size	Raw Size	zfp Size	Total time (s)	Reduction	Throughput
128	25 MB	2.3 MB	.334	x10.9	74.85 MB/s
192	81 MB	8.1 MB	1.05	x10	77.14 MB/s
256	192 MB	18 MB	2.09	x10.7	91.87 MB/s

Table 3. zfp Compression by cube and time

zfp provides the ability to store the *entire* dataset in a lossy compressed mode. Each cube is saved as a VTK Image Data file which uses a few lines of XML for metadata about the object (for example, dimensions and array names), and a VTK float array that is compressed using zfp and saved as binary appended data to the XML file. Figure 6 shows a surface representation of a 256 cube of velocity data. The left figure is the raw velocity magnitude, while the right figure was compressed by zfp and then decompressed for visual representation. The two cubes are indistinguishable in this figure and also when viewing at all zoom levels. Visual equivalence holds when deriving fields of interest from compressed velocity data, including Q-criterion and vorticity magnitude.

5.3 Compressing Dilated Threshold with zfp

We also evaluate using zfp as a compressor for threshold reduced data as compared with VTK's default zlib and we conclude that zlib is preferable; it is lossless and provides comparable compression. The data output from a threshold is sparse and best represented by an irregular grid.

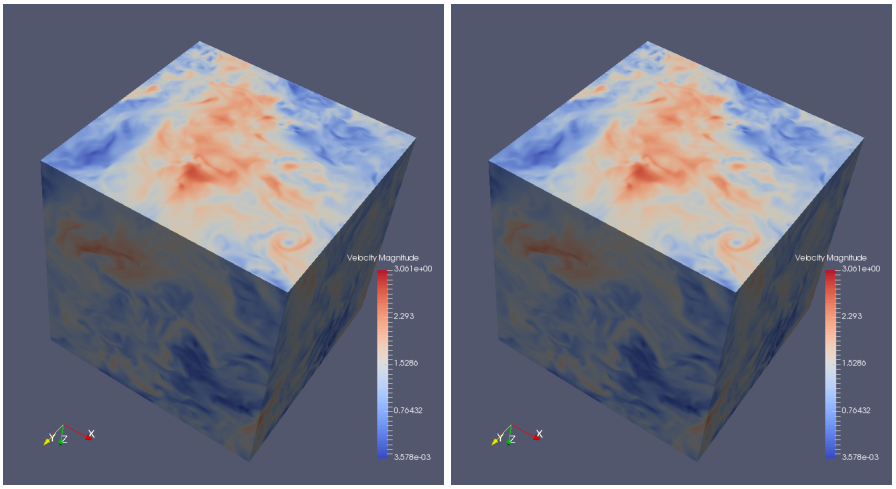

Fig. 6. Isosurface of a 256 cube of isotropic turbulence velocity data. Left: Raw velocity. Right: zfp Compressed at 10^{-1} tolerance.

While zfp does not work on an unstructured grid, version 0.5 added the capability of reducing storage of a block of 4^3 values to one bit if all values within a block are below the error tolerance, since then the block can be approximated as all zeros. To test zfp on sparse data, we performed another experiment to determine if zfp can be used instead of ZLib for our dilated threshold extraction. In this test, we compress a 256^3 block of dilated thresholded data. The data is stored on a structured grid with zero values where the threshold was not met. Utilizing ZLib compression, the result is 6.7 MB, and using zfp it was reduced to 5.8 MB. This compression was performed using a loss threshold of 1×10^{-1}. Using a decreased threshold of 1×10^{-2} resulted in a file size of 8.1 MB. Since the lossy compression did not create a significant reduction in data size, we recommend using a lossless compressor for dilated velocity extraction.

5.4 Multiprocessing Simulation Outputs

We move from microbenchmarks on individual cubes, to the parallel extraction of an entire simulation timestep across many nodes in order to demonstrate that extraction can meet the time constraints of data lifetimes in burst buffers. Our target is to scale these results to the Trinity supercomputer. However, we have to use the development cluster as a proxy. We start by examining the amount of parallelism appropriate for a single burst buffer node. Our treatment examines the amount of parallelism per node to maximize throughput. We initially utilized a node for testing that contained 6 cores and an SSD burst buffer. The results of executing velocity extraction in parallel are shown in Table 4.

The first test labled “Single” in the table displays the times for a single threaded extraction running on a single core. This test was performed in order

to benchmark throughput on a single core. Next we performed the extraction in parallel across six cores. While the overall throughput is the combined speed of all six cores (approximately 93MB/s), the individual throughput per core is less than when the extraction is performed on a single thread. Since I/O is shared on the burst buffer and memory, each core must compete for disk and memory I/O. The data shows that the resultant per core slowdown is about 25% for disk I/O and 20% for Q-Criterion computation. These results informs us that adding more cores to an extraction node will not linearly improve extraction throughput due to memory I/O contention.

Core	Read	Q Comp	Write	Total	Speed
Single	.136	6.55	1.08	9.522	20.17 MB/s
Core 0	.179	8.28	1.81	12.34	15.56 MB/s
Core 1	.179	8.29	1.82	12.37	15.52 MB/s
Core 2	.181	8.37	1.77	12.38	15.51 MB/s
Core 3	.180	8.35	1.79	12.40	15.48 MB/s
Core 4	.183	8.37	1.79	12.41	15.47 MB/s
Core 5	.179	8.15	1.79	12.42	15.46 MB/s
Multi	.180	8.30	1.80	12.39	93 MB/s (total)

Table 4. 256 Cube: Single vs. Multiprocessing I/O by core and averaged (in seconds). Multi throughput is the sum of throughput for all cores.

In the next step of testing, we utilized 32 burst buffer nodes to perform the computation in parallel on the number of cubes required to build an entire 8192^3 timestep. We performed dilated velocity extraction and zfp compression on 32,768 blocks of 256^3 raw velocity data (1024 blocks per node). Figure 7 shows the results of the total extraction time by node and extraction type. Since the Darwin cluster is heterogeneous, the number of cores per node are specified. As evidenced by nodes 27 through 32, four cores were not enough to complete either extraction in under ten minutes. However, nodes 1 through 26 were able to complete each extraction in less than ten minutes.

For in-transit analysis and visualization, the extracted data could be read in place by a viewer such as Paraview to monitor the simulation during the run, and burst buffers provide fast reads in this process. For extracted data that needs to be saved for long term storage, a secondary process copies the results to a shared storage. The extracted data is significantly reduced in size, which also reduces the I/O burden when writing to shared storage. Due to the reduction, the write would not interfere with the simulation.

6 Recommendations for Exascale Simulations

Prior to performing an exascale simulation with the intent to store significant data, it is important to determine what data must be extracted to perform the

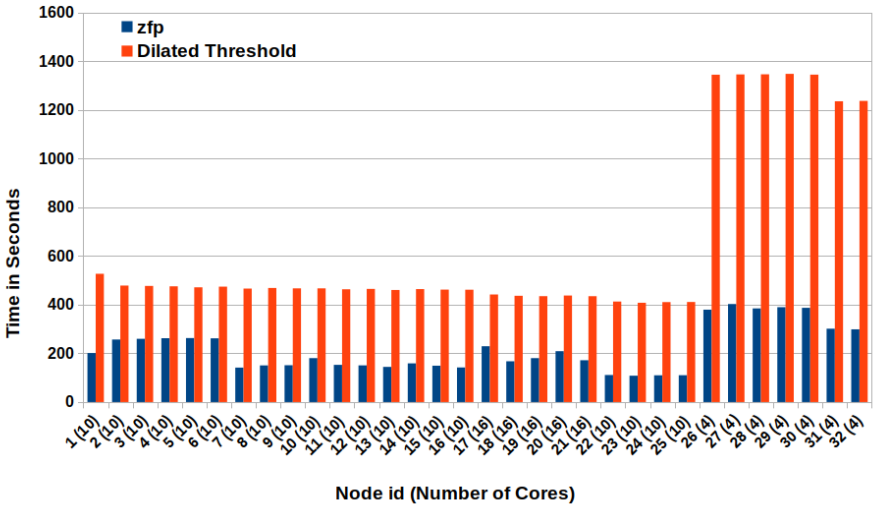

Fig. 7. Times per node for extraction and compression

post scientific analysis. Thus far, we have proposed a method to gather velocity data that contain high vorticity in a scientific dataset, while still capturing a broad view of the overall simulation utilizing a modern lossy data compression algorithm. These two methods combined present a state-of-the-art way to extract useful data from a world class computational fluid dynamics simulation. The lossy compression method with zfp can be used on virtually any scientific dataset that is on a dense structured grid. The compression works optimally on 3D data, but also can work on 2D and 1D floating point datasets. Our extraction techniques present a way forward on how to handle petabyte or even exabyte scale information. The essential part of these exascale simulations is to have a method to extract relevant data utilizing the architecture of modern supercomputers. Having a plan and understanding the data necessary to make scientific discoveries in the future is the key to gathering useful data from an exascale simulation.

7 Conclusion

We have demonstrated the ability to extract scientific data from a world class simulation using burst buffer SSD technology. We demonstrated a two-pronged approach that captures velocity in highly vortical regions along with a lossy compressed representation of the entire velocity dataset for concise storage and future scientific analysis. We demonstrated that we can reduce data by at least an order of magnitude for full field lossy compressed form, and nearly 30x reduction for dilated velocity in high vorticity regions by utilizing the burst-buffer to read raw data and writing extracted and/or compressed data to shared storage.

The extracted data can be utilized for various scientific applications from visualization to tracking highly vortical regions. The lossy compression can be utilized on any dense grid dataset, therefore this method is not limited to turbulence data.

8 Future Work

Since we have outlined and demonstrated the ability to create a meaningful dataset of an exascale simulation, we intend to gather data from an exascale simulation and ingest it into the Johns Hopkins Turbulence Databases a publicly accessible database, providing world-wide access to this compressed and high-Q velocity region dataset. The two-pronged approach will provide scientists the ability to query and perform analysis across the entire exascale simulation result. In addition, we will explore more efficient representations of sparse data. The vtkUnstructuredGrid primitive we utilized stores real space coordinates that result in each velocity vector containing three additional floating point coordinates. A custom representation method that more efficiently represents the points would allow us to even further reduce the space required for this data. In this work we defined a threshold for the simulation, however this may not be optimal in detecting all vortex structures. Our methodology could be expanded to utilize an adaptive threshold for feature extraction as explained in [26]. As scientific needs emerge and new practices for detecting high-vorticity regions are introduced, they could be implemented using our approach to feature extraction.

Acknowledgments

The authors would like to thank Los Alamos National Laboratory for providing compute resources. Specifically we would like to thank Ryan Braithwaite who configured the Darwin cluster and setup our reservation times to run our experiments. This work is supported in part by the National Science Foundation under Grants CMMI-0941530, OCI-108849, ACI-1261715, No. OCI-1244820, and AST-0939767, Johns Hopkins University's Institute for Data Intensive Engineering & Science, Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, and was partially supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, and under the auspices of the U.S. Department of Energy.

References

1. Avinash Sodani. Race to exascale: Opportunities and challenges. *IEEE/ACM International Symposium on Microarchitecture, Micro-44*, 2011.
2. Jason Hick. I/O requirements for exascale. *Open Fabrics Alliance*, 2011.
3. Nicole Hemsoth. Burst buffers flash exascale potential. *HPC wire, 1 May 2014*, 2014.

4. J. Bent, G. Grider, B. Kettering, A. Manzanares, M. McClelland, A. Torres, and A. Torrez. Storage Challenges at Los Alamos National Labs. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–5, April 2012.
5. Yi Li, Eric Perlman, Minping Wan, Yunke Yang, Charles Meneveau, Randal Burns, Shiyi Chen, Alexander Szalay, and Gregory Eyink. A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, 9:N31, 2008.
6. Kai Bürger, Marc Treib, Rüdiger Westermann, Suzanne Werner, Cristian Constantin Lalescu, Alexander S. Szalay, Charles Meneveau, and Gregory L. Eyink. Vortices within vortices: hierarchical nature of vortex tubes in turbulence. *Computing Research Repository*, abs/1210.3325, 2012.
7. G. Eyink, E. Vishniac, C. Lalescu, H. Aluie, K. Kanov, K. Bürger, R. Burns, C. Meneveau, and A. Szalay. Flux-freezing breakdown in high-conductivity magnetohydrodynamic turbulence. *Nature*, 497(7450):466–469, 2013.
8. John Bent, Garth Gibson, Gary Grider, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, and Meghan Wingate. PLFS: A checkpoint filesystem for parallel applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 21:1–21:12, New York, NY, USA, 2009. ACM.
9. ACES Team. Trinity platform introduction and usage model. Los Alamos National Laboratories, number LA-UR-15-26834, 2015.
10. D. H. Ang, M. Brim, S. Parker, G. Watson, and W. Bland. Providing a robust tools landscape for coral machines. In *Workshop on Extreme Scale Programming Tools*, 2015.
11. K. Kanov, C. Lalescu, and R. Burns. Efficient evaluation of threshold queries of derived fields in a numerical simulation database. In *Extending Database Technology*, pages 301–312, 2015.
12. P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, Dec 2014.
13. John Bent, Sorin Faibish, James Ahrens, Gary Grider, John Patchett, Percy Tzelnic, and Jon Woodring. Jitter-free co-processing on a prototype exascale storage stack. In *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–5. IEEE, 2012. LA-UR-pending.
14. Kwan-Liu Ma, Chaoli Wang, Hongfeng Yu, and Anna Tikhonova. In-situ processing and visualization for ultrascale simulations. *Journal of Physics: Conference Series*, 78(1):012043, 2007.
15. J. Ahrens, Li-Ta Lo, B. Nouanesengsy, J. Patchett, and A. McPherson. Petascale visualization: Approaches and initial results. In *Ultrascale Visualization, 2008. UltraVis 2008. Workshop on*, pages 24–28, Nov 2008.
16. F. Chen, M. Flatken, A. Basermann, A. Gerndt, J. Hetheringthon, T. Krüger, G. Matura, and R. W. Nash. Enabling in situ pre- and post-processing for exascale hemodynamic simulations - a co-design study with the sparse geometry Lattice-Boltzmann code HemeLB. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 662–668, Nov 2012.
17. Teng Wang, Kathryn Mohror, Adam Moody, Kento Sato, and Weikuan Yu. An ephemeral burst-buffer file system for scientific applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 69:1–69:12, 2016.

18. N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. On the role of burst buffers in leadership-class storage systems. In *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–11, April 2012.
19. Dong Li, Jeffrey S. Vetter, Gabriel Marin, Collin McCurdy, Cristian Cira, Zhuo Liu, and Weikuan Yu. Identifying opportunities for byte-addressable non-volatile memory in extreme-scale scientific applications. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium*, IPDPS '12, pages 945–956, Washington, DC, USA, 2012. IEEE Computer Society.
20. Katepalli Sreenivasan P. K. Yeung, X. M. Zhai. Extreme events in computational turbulence. *Proceedings of the National Academy of Sciences of the United States of America*, 112(41):12633–12638, 2015.
21. Yves Dubief and Franck Delcayre. On coherent-vortex identification in turbulence. *Journal of Turbulence*, 1:N11, 2000.
22. S. Kida and H. Miura. Identification and analysis of vortical structures. *European Journal of Mechanics - B/Fluids, Volume 17, Issue 4, July-August 1998*, pages 471–488, 1998.
23. Jinhee Jeong and Fazle Hussain. On the identification of a vortex. *Journal of Fluid Mechanics*, 285:69âĂŞ94, 1995.
24. Hendrik Tennekes and John Leask Lumley. *A first course in turbulence*. M.I.T. Press, Cambridge (Mass.), London, 1972.
25. Will Schroeder, Ken Martin, and Bill Lorensen. *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th Edition*. Kitware, 2006.
26. Peer-Timo Bremer, Andrea Gruber, Janine C. Bennett, Attila Gyulassy, Hemanth Kolla, Jacqueline H. Chen, and Ray W. Grout. Identifying turbulent structures through topological segmentation. *Communications in Applied Mathematics and Computational Science*, 11, No. 1:37–53, 2016.