WAVELET COMPRESSION FOR VISUALIZATION AND ANALYSIS ON HIGH PERFORMANCE COMPUTERS

by

SHAOMENG (SAMUEL) LI

A DISSERTATION

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

March 2018

DISSERTATION APPROVAL PAGE

Student: Shaomeng (Samuel) Li

Title: Wavelet Compression for Visualization and Analysis on High Performance Computers

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science by:

Hank Childs	Chair
Allen Malony	Core Member
Boyana Norris	Core Member
Emilie Hooft	Institutional Representative

and

Sara D. Hodges	Interim Vice Provost and Dean of the Graduate School

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded March 2018

© 2018 Shaomeng (Samuel) Li
All rights reserved.

DISSERTATION ABSTRACT

Shaomeng (Samuel) Li

Doctor of Philosophy

Department of Computer and Information Science

March 2018

Title: Wavelet Compression for Visualization and Analysis on High Performance Computers

As HPC systems move towards exascale, the discrepancy between computational power and I/O transfer rate is only growing larger. Lossy in situ compression is a promising solution to address this gap, since it alleviates I/O constraints while still enabling traditional post hoc analysis. This dissertation explores the viability of such a solution with respect to a specific kind of compressor — wavelets. We especially examine three aspects of concern regarding the viability of wavelets: 1) information loss after compression, 2) its capability to fit within in situ constraints, and 3) the compressor's capability to adapt to HPC architectural changes. Findings from this dissertation inform in situ use of wavelet compressors on HPC systems, demonstrate its viabilities, and argue that its viability will only increase as exascale computing becomes a reality.

This dissertation includes previously published co-authored material.

CURRICULUM VITAE

NAME OF AUTHOR: Shaomeng (Samuel) Li

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR, USA
Tufts University, Medford, MA, USA
University of Science and Technology of China (USTC), Hefei, Anhui, China

DEGREES AWARDED:

Doctor of Philosophy, 2018, University of Oregon
Master of Science, 2013, Tufts University
Bachelor of Engineering, 2010, USTC

AREAS OF SPECIAL INTEREST:

High Performance Computing
Scientific Visualization
Data Reduction Techniques

PROFESSIONAL EXPERIENCE:

Software Engineer, National Center for Atmospheric Research (NCAR), Fall 2016 - Present

Graduate Teaching Assistant, University of Oregon, Spring 2016

Graduate Research Assistant, University of Oregon, 2013 - 2016

Summer Intern, Los Alamos National Laboratory, Summer 2016

Visiting Scholar, NCAR, Winter 2016 and Summer 2015

GRANTS, AWARDS AND HONORS:

CISL Special Recognition Award, NCAR, 2017

Advanced Study Program (ASP) Award, NCAR, Winter 2016

Best Paper, Visualization and Data Analysis (VDA) Conference, 2015

PUBLICATIONS:

Li, S., Larsen, M., Clyne, J., & Childs, H. (2017). Performance Impacts of In Situ Wavelet Compression on Scientific Simulations. In *In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV).* ACM.

Li, S., Sane, S., Orf, L., Mininni, P., Clyne, J., & Childs, H. (2017). Spatiotemporal Wavelet Compression for Visualization of Scientific Simulation Data. In *IEEE International Conference on Cluster Computing (CLUSTER).* IEEE.

Baker, A., Xu, H., Hammerling, D., **Li, S.**, & Clyne, J. (2017). Toward a Multi-method Approach: Lossy Data Compression for Climate Simulation Data. In *The 1st International Workshop on Data Reduction for Big Scientific Data (DRBSD-1).* Springer.

Li, S., Marsaglia, N., Chen, V., Sewell, C., Clyne, J., & Childs, H. (2017). Achieving Portable Performance For Wavelet Compression Using Data Parallel Primitives. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV).* The Eurographics Association.

Li, S., Gruchalla, K., Potter, K., Clyne, J., & Childs, H. (2015). Evaluating the Efficacy of Wavelet Configurations on Turbulent-Flow Data. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV).* IEEE.

Li, S., Crouser, R-J., Griffin, G., Gramazio, C., Schulz, H-J., Childs, H., & Chang, R. (2015). Exploring Visualization Designs Using Phylogenetic Trees. In *SPIE Visualization and Data Analysis (VDA).* SPIE.

ACKNOWLEDGEMENTS

I thank my family for the selfless support they provided to my pursue of this degree. They have always encouraged me to follow my dreams and explore the world, even though this meant that their only child stayed abroad for years. I also appreciate my girlfriend Bo very much for her understanding and effort to make our relationship work out while I was pursuing this degree.

I sincerely thank my adviser, Hank Childs, for his countless support and generous encouragement to me. Hank was not only an academic adviser for me, but also a mentor for a young man who was establishing his career in a new country.

I was very fortunate to have two more great mentors, John Clyne and Remco Chang. They provided numerous support and fought opportunities for me, both career and life wise. My life path would have been very different without you.

I had the best co-authors in my career. They helped me not only research better, but also write better (especially in English). They are Matt Larsen, Sudhanshu Sane, Nicole Marsaglia, Vincent Chen, Kristi Potter, Leigh Orf, Pablo Mininni, Christopher Sewell, Kenny Gruchalla, Allison Baker, Haiying Xu, Dorit Hammerling, Jordan Crouser, Garth Griffin, Connor Gramazio, and Hans-Jörg Schulz.

I would also like to thank my committee members: Allen Malony, Boyana Norris, Emilie Hooft, and Hank Childs. Your input steered my research, and thus this dissertation, toward a better shape.

Finally, my fellow friends in the CDUX research group, the Computer Science department, and the University of Oregon campus have made my Ph.D. life colorful; many thanks to all of you.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

LIST OF ALGORITHMS

Algorithm Page

CHAPTER I

INTRODUCTION

1.1 Motivation

Scientific simulations on high performance computing (HPC) systems usually advance and output their states at regular (or irregular) intervals. Each output is effectively a snapshot in time, or "time slice" of what is happening in the simulation. The course of a simulation is then temporally sampled by the output time slices, which are saved in storage, usually parallel filesystems. The policy for when to store a time slice to disk varies by simulation code, with examples such as "every 100^{th} cycle," "every five seconds of simulation time," and "every hour of computation time." Importantly, these time slices have typically been stored at their native resolutions, meaning that the simulation mesh is not modified, and every field value on that mesh is stored (at least for the fields that are stored). With the traditional paradigm for visualizing and analyzing the simulation data, the saved time slices are read out at a later time for visualization and analysis. This paradigm is often referred to as "post hoc" analysis.

As supercomputers get larger and larger, a consistent trend has been that the ability to generate data outpaces the ability to perform I/O, resulting in a *relatively* reduced I/O. An example is the current and next generation supercomputer systems in the Oak Ridge National Lab, where the next generation system (named Summit) will have 5X computational power compared to the current system (named Titan), while its I/O stays almost the same. In response to reduced I/O, there are three main strategies. First, save data less often. As trends worsen, this strategy may become unpalatable for many application domains, since temporal sparsity can result in loss of science. Second, do visualization and analysis

in situ, where the I/O step is completely skipped. This strategy is increasingly being preferred for the cases where domain scientists know what they want to analyze a priori. However, for data exploration-oriented use cases, where new science is often discovered, there often is no a priori knowledge of what to look for. This observation motivates the third strategy, which is to use a combination of in situ and post hoc techniques. In the in situ phase, data is transformed and reduced before saved to storage, with hopes that the reduction will be sufficient to meet I/O requirements. In the post hoc phase, data in the transformed and reduced form is still available for exploration-oriented use cases. That said, the assumption from the traditional paradigm that data be stored at full resolution and all field values are stored, essentially equates to lossless compression, which limits how much reduction can be achieved. As a result, research in this third strategy often assumes that domain scientists will accept lossy techniques when I/O constraints preclude their traditional workflow. This assumption is important, since allowing for some loss in data integrity enables the strategy to be practical.

There are multiple data reduction techniques that fit in the in situ + post hoc strategy; this dissertation specifically focuses on one family of those, namely wavelet-based lossy compression. We also note that there are different approaches to reduce the data size which fit within this strategy, and these approaches are surveyed in Section 3.1.1.

Wavelet-based lossy compression techniques have proven to be effective in the image processing community, such as the latest still image compression standard JPEG2000 (Skodras, Christopoulos, & Ebrahimi, 2001). Its main idea is to exploit the coherence in the data (i.e., neighboring values are likely to have similar values), allowing redundancy to be discarded at a cost of a small accuracy

loss. While image data and scientific data share characteristics, there are still unknowns of applying wavelet compression to scientific data, especially in the intended in situ compression use cases. These unknowns are one of the motivations for this dissertation, which takes the first steps to find the answers.

1.2 Research Goals and Approaches

The research direction of this dissertation is to investigate and better understand the viability of wavelet compression being used as an in situ compressor in an HPC environment. More specifically, this dissertation has three research goals.

First, we aim to find the most suitable wavelet approaches and parameters for compressing scientific data with respect to scientific visualizations. Though one could reasonably assume that the benefits from wavelet compression in image processing will translate, the magnitude of effects are still unknown. We take various established visualizations, tailor evaluation metrics to fit in each one of them, and quantify the impact of information loss to science discoveries. This goal is really twofold: we determine the most suitable parameters through a set of evaluations, at the same time we better understand the efficacy of wavelet compression on real-world visualization and analysis.

Second, we aim to better understand and quantify the performance impacts in situ wavelet compression brings to simulations. Improved I/O is at the center of many aspects of these impacts. For the work described in this dissertation, we tightly couple a wavelet compressor with a proxy simulation so they can run in situ. We perform a weak scaling experiment with up to 1,000 compute nodes, and measure the I/O and computational impacts on a leading edge HPC system.

Third, we aim to develop new approaches for wavelet compression to make use of emerging hardware on cutting-edge HPC systems, namely many-core architectures and solid state drives (SSDs). For many-core architectures, our approach is to utilize a "parallel data primitives" (DPPs) programming paradigm so one implementation can run on multiple architectures without significant performance penalty. We compare the performance of this implementation with architecture specific implementations to demonstration its viability. For SSDs, our approach is to propose and evaluate a new compression scheme, spatiotemporal compression, which is only enabled by the SSDs' capability to temporarily store multiple time slices before processing them. We study the characteristics and viability of this new scheme.

These three research goals combine to inform the big picture on in situ wavelet compression and its potential to address the I/O challenges on HPC systems.

1.3 Dissertation Outline

The dissertation is organized to have either one or two Chapters address one of the above mentioned research goals. Chapter II investigates the wavelet compression approaches and parameters for scientific data through evaluations on real-world visualization and analysis. Chapter III investigates the viability of in situ wavelet compression on a leading edge HPC system. Chapter IV and Chapter V both explore means for wavelet compressors to adapt to emerging architectures, with Chapter IV focusing on many-core architectures and Chapter V focusing on SSDs. Finally, Chapter VI concludes this dissertation and discusses further research directions.

1.4 Co-authored Material

Much of the work in this dissertation is from previous collaborative publications. Below is a listing connecting the chapters with the publications and authors that contributed. A more detailed elaboration on the division of labor is also provided at the beginning of each chapter. That said, for each of these publications, I was not only the first-author of the paper, but also the primary contributor for implementing systems, conducting studies, and writing manuscripts.

- Chapter II is mainly based on publication (Li, Gruchalla, Potter, Clyne, & Childs, 2015), and is a collaboration between Kenny Gruchalla, Kristi Potter, John Clyne, Hank Childs, and myself.

- Chapter III is mainly based on publication (Li, Larsen, Clyne, & Childs, 2017), and is a collaboration between Matt Larsen, John Clyne, Hank Childs, and myself.

- Chapter IV is mainly based on publication (Li, Marsaglia, et al., 2017), and is a collaboration between Nicole Marsaglia, Vincent Chen, Christopher Sewell, John Clyne, Hank Childs, and myself.

- Chapter V is mainly based on publication (Li, Sane, et al., 2017), and is a collaboration between Sudhanshu Sane, Leigh Orf, Pablo Mininni, John Clyne, Hank Childs, and myself.

CHAPTER II

WAVELET COMPRESSION APPROACHES AND THEIR EFFICACY ON SCIENTIFIC DATA

This section contains co-authored material as detailed below. The background (Section 2.1) and wavelet basics (Section 2.2) are compiled from relevant sections of collaborative publications (Li et al., 2015; Li, Larsen, et al., 2017; Li, Marsaglia, et al., 2017; Li, Sane, et al., 2017). Experiments in this chapter (Section 2.4 through 2.7) are specifically from work originally appearing at the IEEE Symposium on Large Data Analysis and Visualization (Li et al., 2015), where Hank Childs provided guidance for the first analysis (Section 2.5.1) and Kenny Gruchalla provided guidance for the second analysis (Section 2.5.2). Further, John Clyne was extremely helpful in assisting with understanding of wavelets. Hank Childs and Kristi Potter were heavily involved in developing the experiment methodology, as well as editing the final paper.

This chapter describes the basics of wavelet transforms and various parameter options to use them to achieve compression. We then apply wavelet compression on two real-world analyses to evaluate the pros and cons of each option, and identify the best ones for scientific data compression. These evaluations serve a twofold purpose, as they also inform the efficacy of wavelet compression with respect to real-world analyses. We note that though studies in this chapter are complete by themselves, many experiments throughout the dissertation also provide insights into the efficacy of wavelets with different settings. We briefly summarize these experiments towards the end of this chapter in Section 2.8.

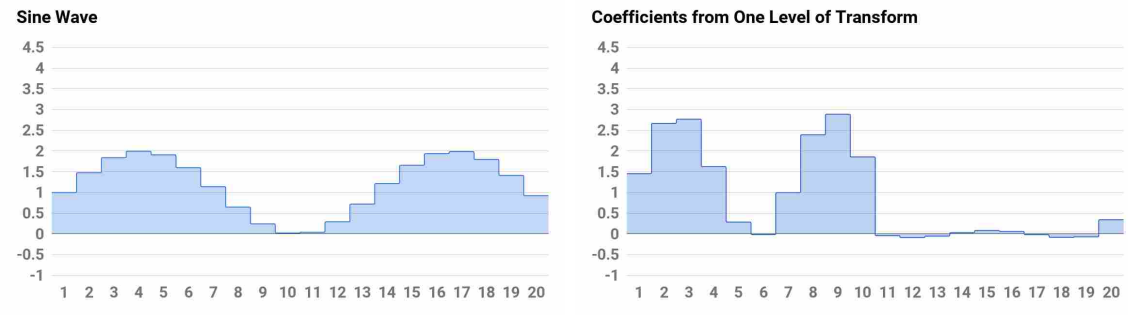


Figure 1. Demonstration of information compaction of wavelets. The input is a discrete sine wave with 20 data points (left), and the output is 20 wavelet coefficients (right). The magnitude of the wavelet coefficients is proportional to their information content.

2.1 Background

There are multiple approaches to use wavelets for compression, and at the heart of each approach is *wavelet transform*. Wavelet transforms operate similarly to Fourier transforms in that they both represent data in another domain. In contrast to Fourier transforms which represent data in the frequency domain, wavelet transforms represent data in the wavelet domain, which contains both frequency and time information.

The wavelet domain representation has a few desirable properties that are used in a wide range of applications, with one being lossy compression. Compression is enabled by the *information compaction* property of wavelets. More specifically, wavelet transforms are able to remove coherence from data so most of the information is kept in a small number of coefficients in the wavelets domain. Lossy compression is achieved by discarding the majority of less information-rich coefficients. The more coherence that exists in the data, the better wavelets compacts information, i.e., more information per byte. Figure 1 visualizes the effect of applying wavelet transforms on a sine wave. Here the resulting wavelet coefficients have information content proportional to their magnitude. As the

visualization shows, much information is compacted to a few large magnitude coefficients, resulting in many near-zero coefficients, which are opportunities for lossy compression.

2.2 Basics of Wavelet Transforms

This section provides an abbreviated description of the wavelet transform from the standpoint of compression applications. Excellent and more detailed introductory descriptions are available in (Stollnitz, DeRose, & Salesin, 1996) and (Burrus, Gopinath, Guo, Odegard, & Selesnick, 1998).

2.2.1 One-dimensional Wavelet Transform. It is often advantageous to express a function or signal $x(t)$ as a linear expansion about a set of basis functions. In the case of wavelet bases, such a decomposition is given by:

$$x(t) = \sum_{k\in\mathbb{Z}} \sum_{j\in\mathbb{Z}} a_{j,k}\psi_{j,k}(t), \tag{2.1}$$

where $a_{j,k}$ are real-valued coefficients, and $\psi_{j,k}(t)$ are wavelet functions typically forming an orthonormal basis. Conceptually, the coefficients, $a_{j,k}$ measure the similarity between the input signal and the basis functions. For finite, discrete $x[n]$ under a "non-expansive" wavelet transform, the number of coefficients $a_{j,k}$ has the same size as $x[n]$. If $x[n]$ exhibits sufficient coherence, and if $\psi_{j,k}(t)$ is suitably chosen, then many of the $a_{j,k}$ coefficients will tend toward zero, with only a fraction of the remaining non-zero coefficients containing most of the energy or information content of the signal.

The resulting wavelet coefficients have two flavors: the "approximation" coefficients which provide a coarsened representation of the data, and the "detail" coefficients which contain the missing information from the coarsened representation. Figure 1 illustrates this intuition: the first ten coefficients

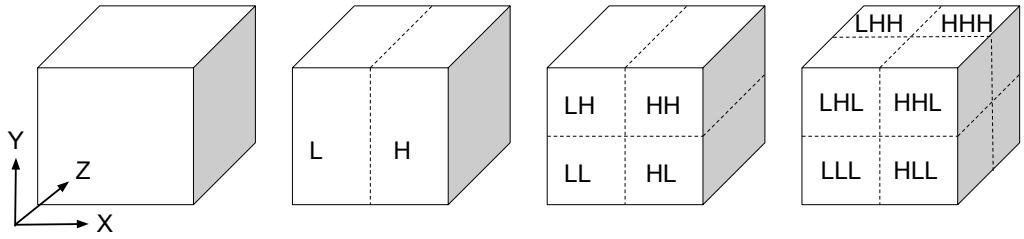


Figure 2. Illustration of one level of wavelet transforms for a three-dimensional cube. Approximation and detail coefficients are denoted using "L" and "H," respectively. From left to right are the original cube, and the resulting coefficients after wavelet transforms in the X, Y, and Z axes, respectively.

approximates the data, and the second ten coefficients provide the deviation information.

Wavelet transforms can then be applied recursively on coefficients from previous wavelet transforms, resulting in a hierarchy of coefficients. This recursive application of wavelet transforms helps concentrate information content into fewer and fewer coefficients, and the resulting coefficient hierarchy enables a data representation spanning multiple resolutions. In Equation 2.1, j indicates the different scales.

2.2.2 Multi-dimensional Wavelet Transforms. The one-dimensional wavelet transform can be extended to multiple dimensions by successively applying a one-dimensional transform along each axis, i.e., output coefficients of one transform become the input of the next transform along a different axis. This practice takes advantage of data coherence along all dimensions.

The ordering of axes to apply transforms on may differ though. Figure 2 illustrates a commonly adopted ordering, referred to as "non-standard decomposition" in some literature. First, each row goes through a wavelet transform pass in the X direction, resulting in approximation and detail coefficients with respect to the X axis. Second, these coefficients then go through wavelet

transforms in the Y direction as columns, resulting in approximation and detail coefficients with respect to the Y axis. Third, the output coefficients from the second set of transforms go through wavelet transforms in the Z direction, resulting in approximation and detail coefficients with respect to the Z axis. Though named "non-standard decomposition," this ordering is actually the most popular one in practice. More details and discussions on the ordering topic could be found in (Stollnitz et al., 1996) and (Burrus et al., 1998).

2.2.3 Wavelet Kernel Choices. Wavelet kernels are used to generate the basis functions from Equation 2.1. These basis functions, in turn, have different efficacies when used for data compression. We consider a basic yet popular wavelet kernel, the Haar kernel, and two more complicated ones, which are both members from the Cohen-Daubechies-Feauveau (CDF) family.

2.2.3.1 Haar Kernel. The Haar (Haar, 1910) kernel is one of the most basic and widely understood wavelet kernels. It serves as a baseline for our evaluation because of its popularity. The Haar kernel generates a series of "square-shaped" functions for its basis functions. When used with the multi-resolution strategy, the Haar kernel yields a hierarchical representation that is identical to that produced with a linear (trilinear in three dimensions) down-sampling filter. Computation wise, the Haar kernel introduces only a modest computational cost.

2.2.3.2 CDF Wavelet Family. The Cohen-Daubechies-Feauveau (Cohen, Daubechies, & Feauveau, 1992) wavelet family has multiple members; they differ based on their filter shapes and sizes. The filter sizes are also used to indicate individual kernel members. For example, CDF 9/7 is a member with filters (for different tasks) of size nine or seven, and CDF 8/4 is a member with filters (for different tasks) of size eight or four. These two are also included in our study.

The CDF family of wavelets is widely used in the compression of non-periodic signals (e.g., images and video) due to its effective boundary handling capabilities (Usevitch, 2001). The CDF 9/7 kernel, in particular, has been empirically shown to yield superior compression distortion results for imagery (Antonini, Barlaud, Mathieu, & Daubechies, 1992; Villasenor, Belzer, & Liao, 1995), and is what is used in JPEG 2000 (Skodras et al., 2001).

All these three wavelet kernels have the property that their wavelet coefficient magnitudes are proportional to their information content. Thus, a sensible approach to compression using wavelet transforms is to retain only coefficients with the largest magnitudes. We will discuss two of these approaches in the following subsection.

2.2.4 Compression Strategy Options. There are multiple ways to lay out wavelet coefficients. Given a storage budget, the ordering of coefficients determines which coefficients are included to reconstruct the approximation, and thus determines the compression strategy. There are two popular strategies: multi-resolution and coefficient prioritization.

2.2.4.1 Multi-resolution. With a multi-resolution approach, coefficients are laid out naturally with respect to the coefficient hierarchy. Because each level of coefficients reconstructs an approximation of the original data array, compression is achieved by storing only some levels of coefficients from the hierarchy. Coefficients stored in this manner retain their addresses, i.e., where they belong to in the coefficient hierarchy, and thus do not require additional addressing mechanisms.

Each iteration of wavelet transform coarsens the data array into half of its previous resolution. As a result, the multi-resolution compression strategy

offers a pyramid representation that is strictly limited to power-of-two reductions along each axis. In the case of a three-dimensional regular mesh, the applicable compression ratios are of the form 8^N:1, where N is the number of iterations of wavelet transform to apply. That is, applicable compression ratios include 8:1, 64:1, 512:1, etc.

The multi-resolution wavelet approach differs from similar techniques used by the visualization community, such as mipmapping (Williams, 1983), and space-filling curves (Morton, 1966) and (Liang, Chen, Huang, & Liu, 2008). In the first case, mipmapping requires additional storage space for the coarsened approximations. In the second case, the coarsened approximations come from single point data (nearest neighbor sampling), rather than average of all points in that region.

2.2.4.2 ***Coefficient Prioritization.*** When reconstructing the original data from the wavelet expansion given in Equation 2.1, coefficients have different importance, i.e., coefficients representing the more rapidly changing parts of the original data contribute more than coefficients representing the more self-similar parts. The prioritized coefficient technique makes use of this property by laying out the coefficients based on their importance, i.e., important coefficients are placed toward the beginning of the storage space. Compression is thus achieved by storing only the collection of important coefficients, and treating the rest of the coefficients as zeros.

The prioritized coefficient strategy differs from the multi-resolution strategy in that: 1) it supports an arbitrary compression ratio, by choosing what percentage of total coefficients to keep; 2) it supports reconstructing the mesh on its full resolution, by assigning zeros to the coefficient locations that are not stored; and

3) it requires extra mechanisms to keep track of where the prioritized coefficients belong in the coefficient hierarchy. One mechanism is to keep coefficient addresses explicitly, thus introducing storage overhead.

We note that more efficient coding and storage of these coefficients is a sizable area of research that we do not address here. Interested readers can consult other sources for SPECK (Islam & Pearlman, 1998), SPIHT (Kim & Pearlman, 1997), and EBCOT (Taubman, 2000), for example, and their corresponding high-dimensional derivatives.

2.3 Prior Research of Wavelets in Visual Analytics

Wavelet compression has been previously employed with scientific visualization. When reconstructing slices of a CT data set, two-dimensional wavelet transforms have been proven to provide high compression rates with fast decoding for performing random access of voxels (Ihm & Park, 1999; Rodler, 1999). When used on three-dimensional volume data sets, such as hydrodynamic simulations, global ocean models, or terrain data, wavelet compression has been shown to be effective when visualizing different levels of detail (Bertram, Duchaineau, Hamann, & Joy, 2000; Olanda, Pérez, Orduña, & Rueda, 2014; Sakai, Sasaki, Obayashi, & Nakahashi, 2013; H. Tao & Moorhead, 1994). Wavelet compression also brings new possibilities for real-time analysis on large-scale data sets on commodity hardware (Gioia, Aubault, & Bouville, 2004; Guthe & Straßer, 2001).

In contrast to previous studies, we consider a variety of wavelet configurations, evaluate tradeoffs in accuracy, storage cost, and execution time, and decide the best parameters for scientific data compression.

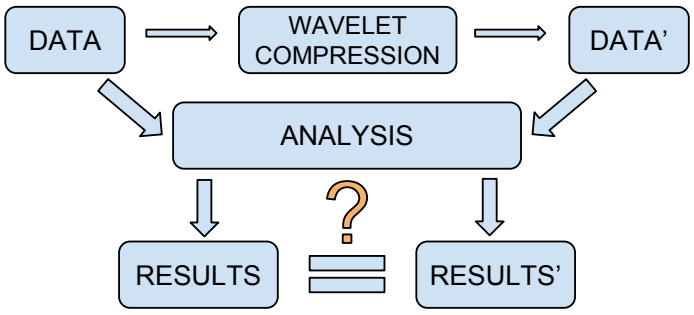


Figure 3. Our experiment methodology. We used wavelet compression to create a compressed form (DATA') from its original form (DATA). *RESULTS* and *RESULTS'* represent the analysis results from *DATA* and *DATA'*, respectively. Our study then quantitatively evaluated the difference between *RESULTS* and *RESULTS'*.

2.4 Study Overview

We studied multiple wavelet configurations, varying over compression strategies, wavelet kernels, and compression ratios. Section 2.4.1 describes our experiment methodology for a generic configuration, and Section 2.4.2 describes the different wavelet configurations we studied.

2.4.1 Experiment Methodology. Our experiment methodology, illustrated in Figure 3, was as follows:

- We began with turbulent flow data in its raw form.
- We applied wavelet compression to the raw data to get the compressed form.
- We applied an analysis routine to the data in both its raw and compressed forms.
- We quantitatively evaluated the difference between the results.

This wavelet transformations were performed using the VAPOR software package (Clyne, Mininni, Norton, & Rast, 2007; Clyne & Rast, 2005). VAPOR also has advantages over other implementations, for example the work by Woodring et al. (Woodring, Mniszewski, Brislawn, DeMarle, & Ahrens, 2011), that VAPOR

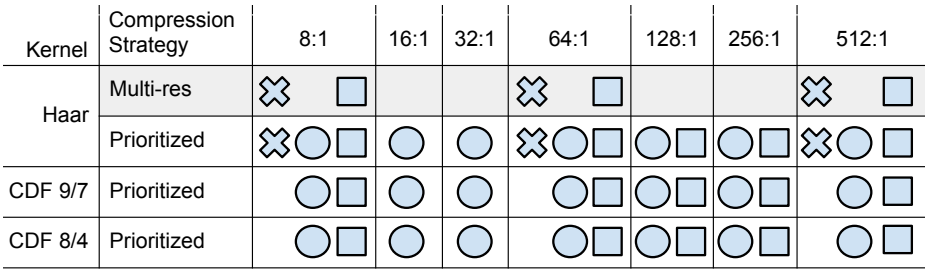


Figure 4. Wavelet configurations studied. Cross signs represent configurations examined in our first round of experiments, circles represent configurations examined in our second round of experiments, and squares represent configurations examined in our third round of experiments.

natively supports wavelet transforms in three dimensions, and operates on floating point data. For the analysis routines, we used the software that was used to perform the analysis originally: VisIt (Childs et al., 2012) or VAPOR.

2.4.2 Wavelet Configurations Studied. We performed our experiments in three rounds.

In the first round, we considered the wavelet compression strategy. Specifically, we compared multi-resolution with prioritized coefficients (see Section 2.2.4), both using the Haar kernel. Since the multi-resolution approach requires two-to-one reduction in all three dimensions, only reductions that are powers of eight are possible. The three compression ratios we studied for this round were 8:1, 64:1, and 512:1.

In the second round, we studied the effects of the wavelet kernels. Specifically, we compared the Haar kernel, the CDF 9/7 kernel, and the CDF 8/4 kernel. All tests in this round used the prioritized coefficients strategy. The compression ratios for this round were 8:1, 16:1, 32:1, 64:1, 128:1, 256:1, and 512:1. With prioritized coefficients, arbitrary ratios would have been possible, but we chose powers of two for easy comparisons with the multi-resolution approach.

In the third round, we repeated the wavelet configurations from the first two rounds, but with a different analysis routine.

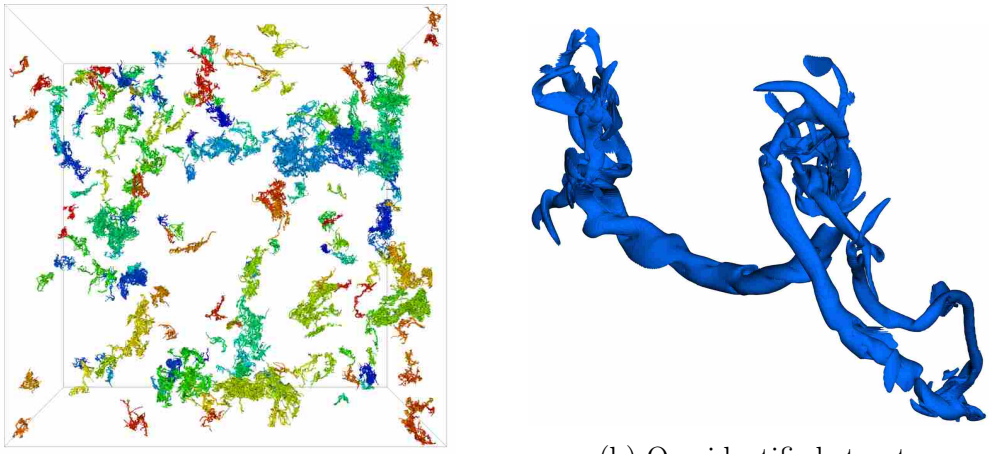
Figure 4 summarizes the wavelet configurations we studied over all three rounds.

2.5 Visual Analytics Overview

Our study incorporated two analysis routines, both of which came from established research on turbulent-flow data. These two analyses were both performed on rectilinear data sets from simulations. In our study, we located the original data sets used in the two predecessor studies. We denote their data sets DS1 and DS2 for simplicity. DS1 contained a single scalar field defined over thirteen time slices on a $4,096^3$ mesh. DS2 did not vary in time. It contained a scalar field and a vector field, both defined on a $1,024^3$ mesh. For DS1 and DS2, the scalar field was "enstrophy," which directly measures the kinetic energy in a flow model. For DS2, the vector field was velocity.

Both established analysis routines identified and analyzed critical structures, which are defined as regions with significantly higher enstrophy values than the areas surrounding it. However, they focused on the critical structures' properties in different scopes. The first analysis focused on the *global population* of all critical structures, while the second analysis focused on the *local dynamics* of individual critical structures.

Finally, we note that our goal is to evaluate tradeoffs in compression and accuracy on established analysis routines with wavelet compressed data over a variety of wavelet configurations. In particular, if an analysis routine is especially sensitive to lost accuracy in the data, then we view that as a finding for our study, but not as a cue that we should extend or modify the established analyses.



(a) The baseline rendering

(b) One identified structure

Figure 5. The left rendering shows critical structures identified from the first time slice of DS1. Each structure has a unique color in this view. The right rendering shows a close-up view of one of the critical structures.

We describe the key steps of these two visual analytics routines in the following subsections.

2.5.1 Critical Structure Identification. Gaither et al. performed an analysis that included measuring the global population of critical structures (Gaither et al., 2012). Identifying these critical structures took two steps. The first step isolated regions with enstrophy values higher than α, a fixed value provided by domain scientists. For reference, the test data set DS1 contains millions of these high-enstrophy regions. The second step eliminated structures with a volume smaller than a threshold β, again a fixed value provided by domain scientists. For DS1, this process reduces the number of critical structures down to hundreds. Figure 5 shows a screenshot of these identified critical structures at the first time slice of DS1, as well as a close-up look at one of the critical structures.

This analysis routine can potentially be quite sensitive to changes in the enstrophy field from compression. If the compressed enstrophy breaks a

component, then the result may put that component below β. Similarly, if the compressed enstrophy joins two disjoint components, then the result may put the joined component above β. Such a change would affect the statistics of the global population of critical structures.

2.5.2 Local Dynamics Analysis. Gruchalla et al. analyzed the dynamics of individual structures in a turbulent-flow simulation (Gruchalla, Rast, Bradley, Clyne, & Mininni, 2009). Specifically, for a single structure, they studied the change in velocity from the inside of the structure to the outside. For their analysis, structures were first identified following steps similar to those described in Section 2.5.1, and then representatives from two distinct types of local dynamics were picked to perform further analysis. Renderings of these two representations can be found in Figure 10a and 10b in the results discussion. The high-enstrophy areas of these structures are rendered in blue, and their local dynamics are illustrated by yellow streamlines seeded in the velocity field. We refer to these two types of structures as S1 and S2. Visually, the two types of dynamics can be distinguished from each other, since streamlines twist around the core with S1, and follow the writhe of the tube with S2.

Their primary analysis looked at *radial-enstrophy profiles* — enstrophy as a function of radius — for individual structures. They created this profile by considering fifteen cross sections along the major axis of a structure. Within a cross section, they identified the center of the structure for that cross section. They then calculated the average enstrophy around this center for many different radii. This resulted in a radial-enstrophy profile for that cross section. They then averaged the radial-enstrophy profiles over all cross sections to create the final radial-enstrophy

profile. Figure 10e and 10f plots the two radial-enstrophy profiles for structures S1 and S2.

A radial-enstrophy profile captures local flow dynamics. For S1, the profile shows high enstrophy values in the core, and drops rapidly when exiting the structure. For S2, the profile shows moderate enstrophy values in the core and drops slowly when exiting the structure.

2.6 Efficacy Evaluations

In this section we present evaluation results from our two established analyses. The first subsection presents the results from the first analysis: critical structure identification. This subsection consists of the first two rounds of our experiment. The second subsection presents the results from the second analysis: local dynamics analysis. This section consists of the third round of our experiment.

For each of the two analyses, we first describe our evaluation metric, and then present the evaluation results.

2.6.1 Evaluation: Critical Structure Identification.

2.6.1.1 Evaluation Metric. The critical structure identification task yields some number of identified structures on both the raw data and the wavelet-compressed data. In the ideal case, the number of critical structures for both would be the same, and each critical structure in the baseline analysis would have a corresponding structure in the same location in the compressed data. However, in practice, the critical structures do not always align in this ideal way.

There are two types of error that can occur. First, a critical structure can appear in the compressed data that does not appear in the raw data. We refer to this type of error as a *false positive*. Second, a critical structure can fail to appear in the compressed data, even though it does appear in the raw data. We refer to

this type of error as a *false negative*. Our evaluation metric is based on the number of false positives and false negatives — the lower these two numbers are, the better the results from the compressed data match the baseline results from the raw data.

To provide a better comparison among all time slices, we considered the proportion of error among the critical structures, rather than absolute numbers. Formally, let FN be the number of false negatives, FP be the number of false positives, and $COMM$ be the number of critical structures common to both. We then focused on two error metrics:

$$FN\_Proportion = \frac{FN}{FN + COMM},$$

and

$$FP\_Proportion = \frac{FP}{FP + COMM}.$$

Both metrics range between zero and one, with values closer to zero being better.

Determining if an identified critical structure is common to both is not a trivial task. We used a proximity test to match up critical structures in $COMM$. This test compared the bounding boxes of all structures in the baseline with the bounding boxes of all structures in the compressed data. For each pair of baseline-and-compressed structures, the overlap was measured. The overlap was calculated so that structures with similar sizes and similar spatial extents would have high values. Specifically, if V was the volume of intersection between the two, V_B was the volume of the baseline structure, and V_C was the volume of the compressed structure, then their overlap was scored as

$$\frac{V^2}{(V_B \times V_C)}.$$

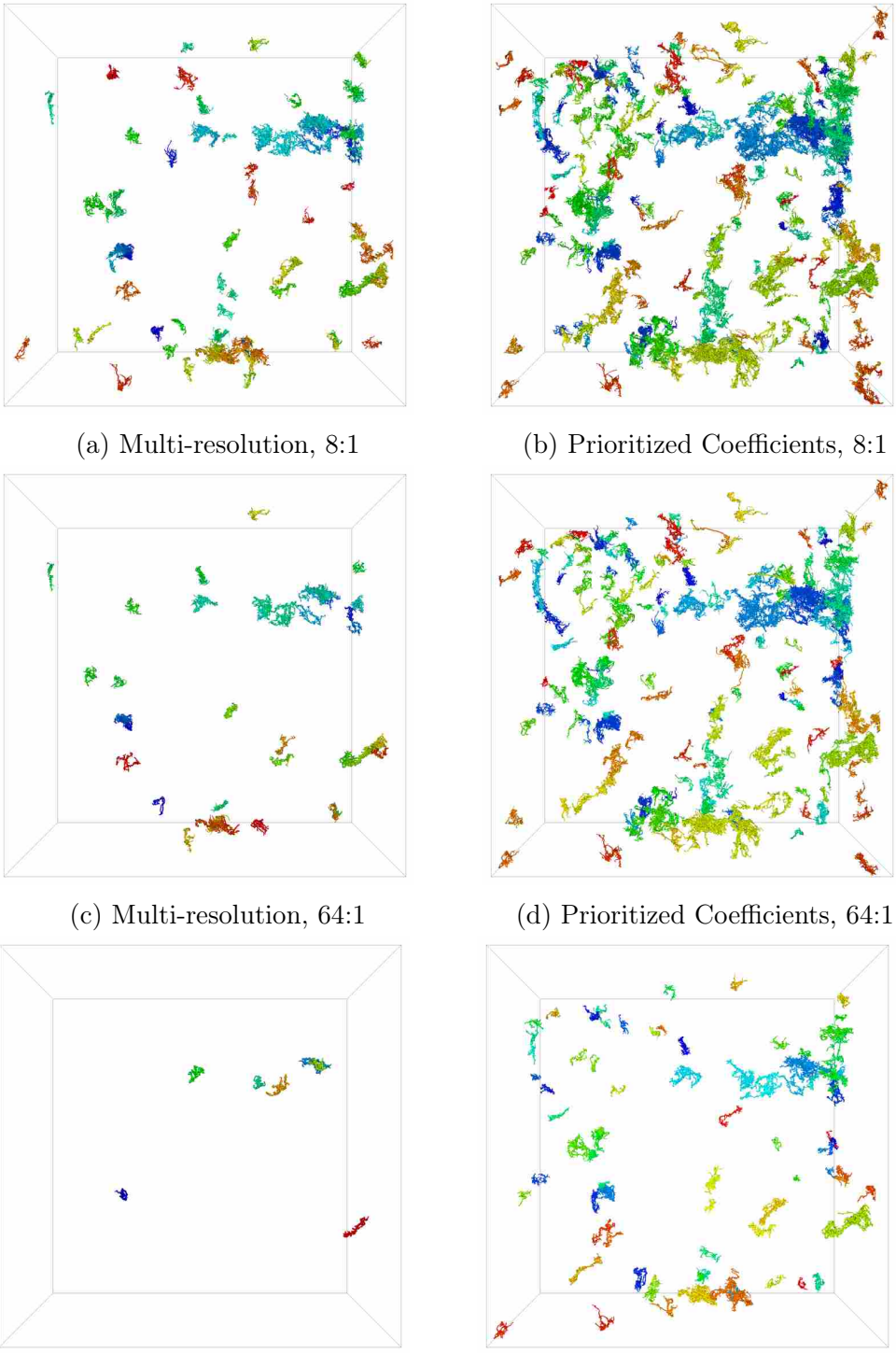
A perfect overlap would score one, and no overlap would score zero. A baseline-compressed structure pair was then identified as the "same" if, for a baseline

structure b and a compressed structure c, then b's best match (i.e., highest score) was c, and c's best match was b. This meant that large baseline structures that got split during compression would contribute false positives (as only one compressed structure would match, but one would find no match), and separate baseline structures that got combined during compression would contribute false negatives (as only one baseline structure would match the combined structure).

2.6.1.2 Results: Wavelet Compression Strategy. We evaluated the two wavelet compression strategies — prioritized coefficients and multi-resolution (see Section 2.2.4) — with three compression ratios: 8:1, 64:1, and 512:1. Each test used the Haar kernel. This resulted in six different wavelet settings. Figure 6 shows renderings from our analysis using each of the six settings on the first time slice of DS1. Visual inspection shows that results using the prioritized coefficient strategy not only retain more critical structures from the baseline, but also preserve more shape details.

Figure 7 compares *FN_Proportion* and *FP_Proportion* between the prioritized coefficient strategy and the multi-resolution strategy. It plots the average values over all thirteen time slices. Prioritized coefficients clearly outperform the multi-resolution approach, as the blue lines are significantly lower than the red ones. That said, FN for multi-resolution drops at the 512:1 ratio. This is because the multi-resolution strategy fails to identify most of the structures at this compression level, so the few identified ones are likely to be correct. Restated, this low false positive proportion does not indicate better performance for the multi-resolution strategy.

2.6.1.3 Results: Wavelet Kernel Choice. We then expanded our kernel choices to include the CDF 9/7 and CDF 8/4 kernels. This meant there



(a) Multi-resolution, 8:1

(b) Prioritized Coefficients, 8:1

(c) Multi-resolution, 64:1

(d) Prioritized Coefficients, 64:1

(e) Multi-resolution, 512:1

(f) Prioritized Coefficients, 512:1

Figure 6. Screenshots from our critical structure identification task using the multi-resolution strategy (left column) and the prioritized coefficient strategy (right column). These screenshots are from the first time slice of DS1, so Figure 5b shows the baseline result for raw data for this task.

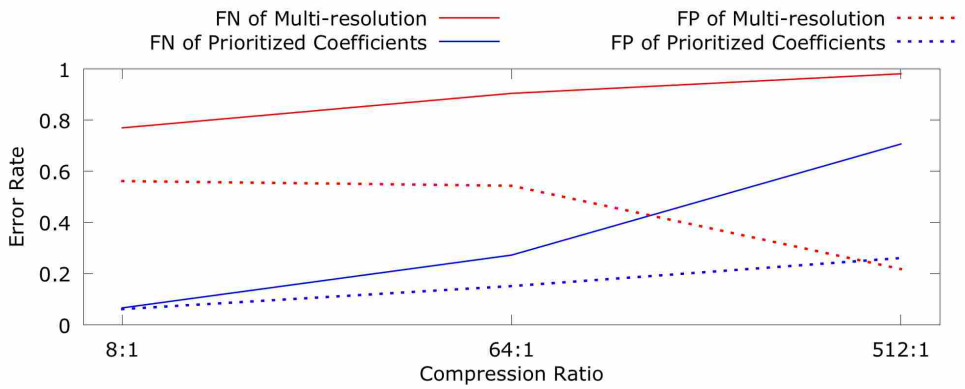
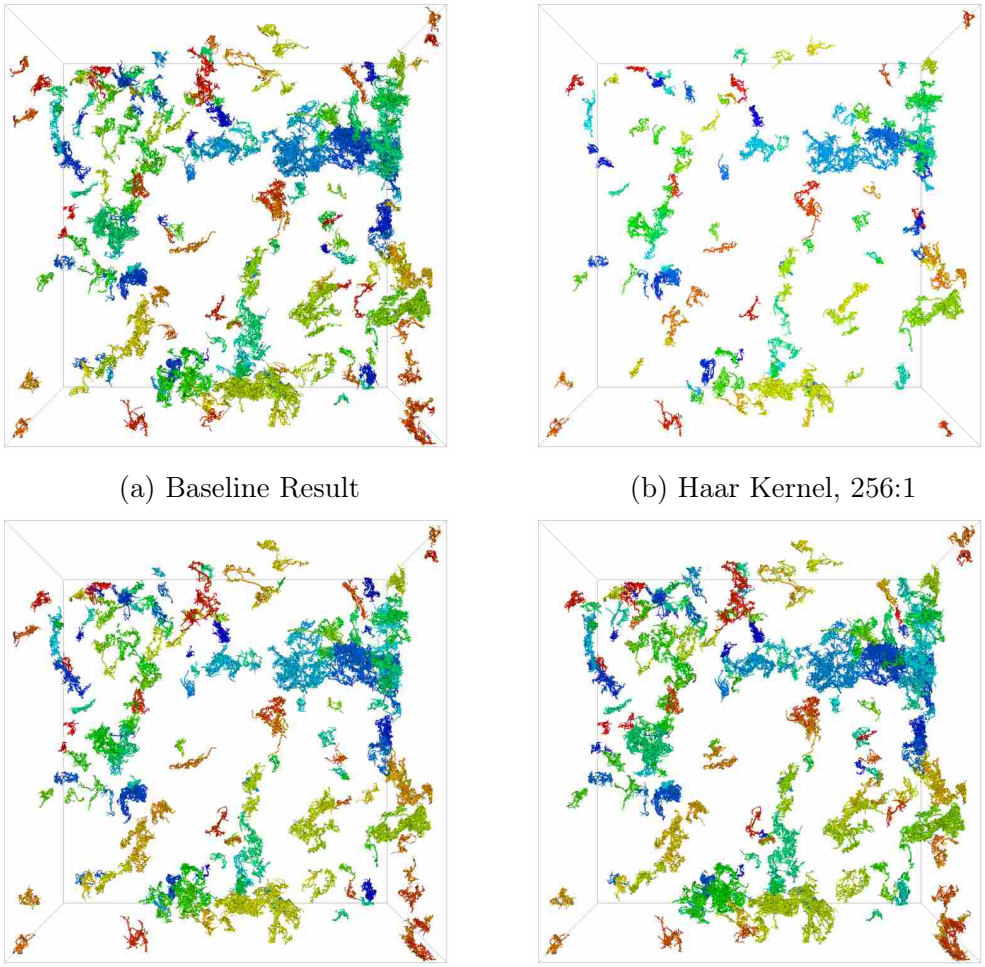


Figure 7. False negative (solid lines) and false positive (dashed lines) proportions for two compression strategies. The multi-resolution results are colored red, and the prioritized coefficient results are colored blue. Each line is an average of results from all thirteen time slices.

were a total of three kernels, as we still considered the Haar kernel. We no longer considered a multi-resolution approach, and this allowed us to consider more compression ratios. We studied seven: 8:1, 16:1, 32:1, 64:1, 128:1, 256:1, and 512:1. Thus, the total number of experiments was 21. Figure 8 shows the visual difference among three wavelet kernels using the 256:1 compression ratio (8b, 8c, and 8d), and their comparison to the baseline (8a). Visual inspection shows that while all kernels capture many structures, CDF 9/7 and CDF 8/4 manage to keep more details than Haar.

We plotted *FN_Proportion* and *FP_Proportion* for the three wavelet kernels in Figure 9. Again they are averaged over all thirteen time slices. The top plot shows that the CDF 9/7 and CDF 8/4 have similar false negative proportions, and they are both lower than the Haar kernel. The bottom plot shows that the CDF 9/7 kernel has the lowest false positive proportions at every compression ratio by a clear margin. Summing up, results from this evaluation indicate that CDF 9/7 is the best choice among these three wavelet kernels for this analysis.

2.6.2 Evaluation: Local Dynamics Analysis.



(a) Baseline Result

(b) Haar Kernel, 256:1

(c) CDF 9/7 Kernel, 256:1

(d) CDF 8/4 Kernel, 256:1

Figure 8. Screenshots from our critical structure identification analysis on the first time slice of DS1. All the compressed results (b, c, and d) use a prioritized coefficient strategy.

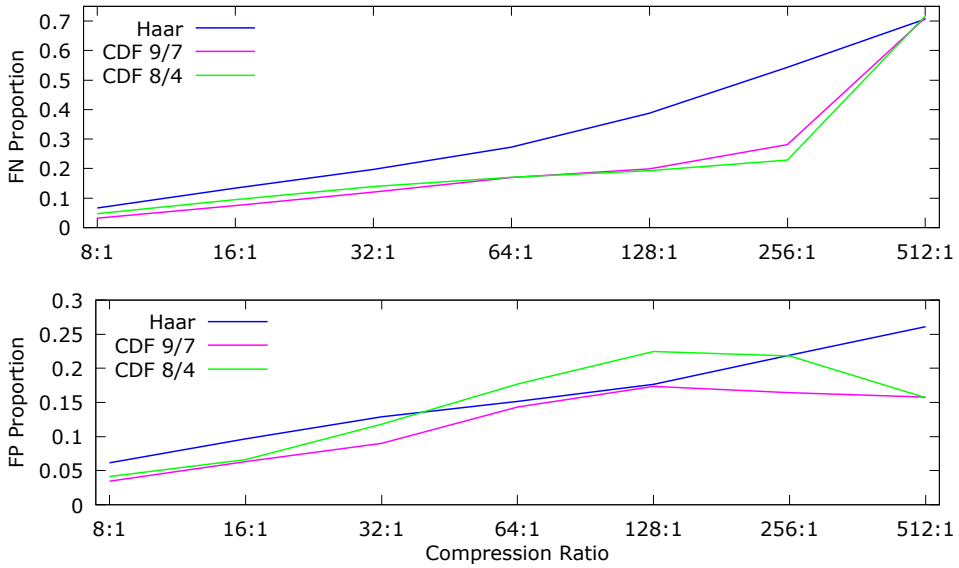
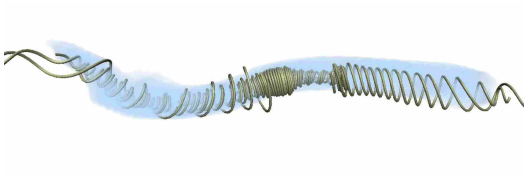


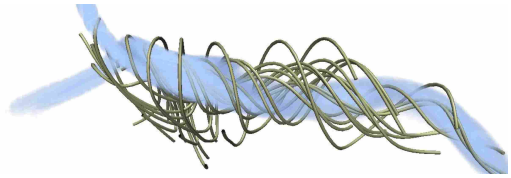
Figure 9. False negative (top) and false positive (bottom) proportions for three wavelet kernels. Each line is an average of results from all thirteen time slices.

2.6.2.1 Evaluation Metric. Our evaluation process began by identifying the two structures, S1 and S2, in the raw and wavelet-compressed versions of the data. We then calculated their radial-enstrophy profiles. Ideally, the profile produced using the compressed data would be the same as the profile from raw data, i.e., the radial-enstrophy plots would overlap with each other. However, in practice, there were differences between the two profile lines. We evaluated the wavelet compression by quantifying the difference between these two radial-enstrophy profiles — the smaller the difference, the better the compressed data preserved local dynamics.

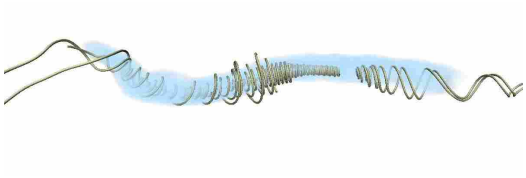
We used the root mean square error (RMSE) metric to measure the difference between the two profiles. Specifically, given the baseline radial-enstrophy profile $E[r]$ $(0 \leqslant r < N)$ and the radial-enstrophy profile from compressed data



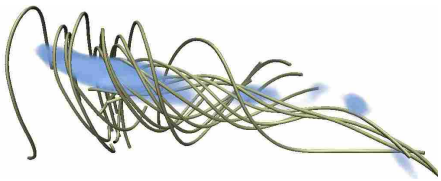
(a) S1: Baseline Rendering



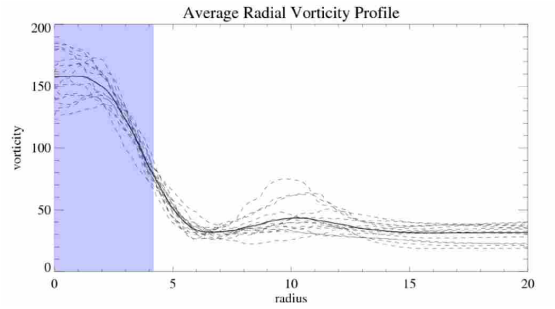
(b) S2: Baseline Rendering



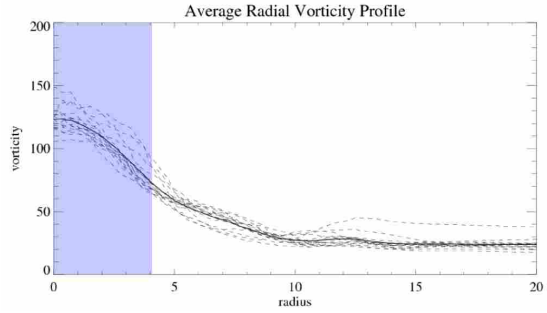
(c) S1: CDF 9/7+Prioritized, 128:1



(d) S2: Haar+Multi-resolution, 64:1



(e) Radial-enstrophy profile for S1, using baseline data.



(f) Radial-enstrophy profile for S2, using baseline data.

Figure 10. Visualizations of identified critical structures (rendered in blue) and their local dynamics (rendered in yellow). The top-row subfigures show the baseline rendering using the raw data; the middle-row subfigures show the rendering using the compressed data; and the bottom-row subfigures show the radial-enstrophy profiles for these two structures.

$\widetilde{E}[r]$ $(0 \leqslant r < N)$, RMSE is then defined as:

$$RMSE \;=\; \sqrt{\frac{\sum_{r=0}^{N-1} \left(E[r] \;-\; \widetilde{E}[r]\right)^2}{N}}\;. \tag{2.2}$$

In this work, we normalized RMSE by the observed data range. Thus, the normalized RMSE (a.k.a. NRMSE) evaluated to zero when $E[r]$ and $\widetilde{E}[r]$ are exactly the same, and evaluated to one in the worse possible case.

2.6.2.2 Evaluation Results. We included all three wavelet kernels and both compression strategies for this evaluation. The multi-resolution strategy

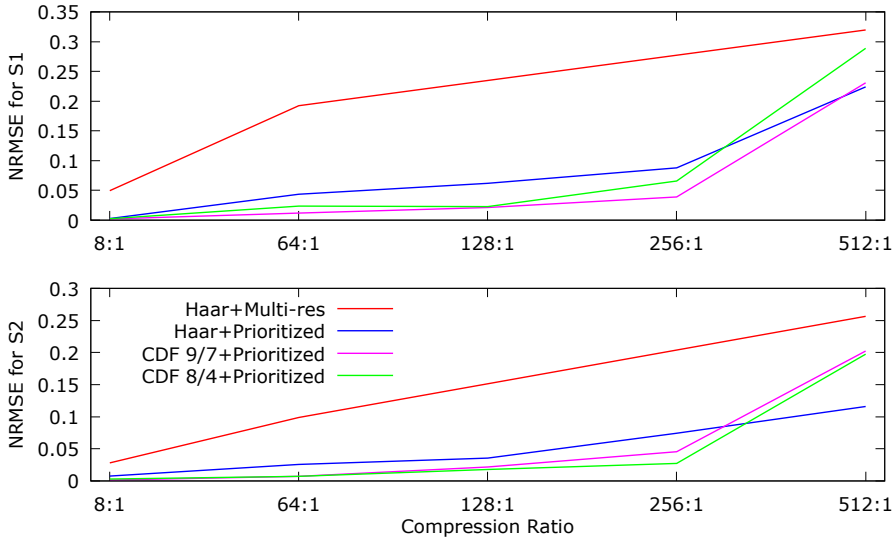


Figure 11. NRMSE of the radial-enstrophy profiles for S1 (top) and S2 (bottom). Each color represents a wavelet setting in our experiment, with the legend displayed in the bottom figure. The red color represents Haar+multi-resolution, which only supports compression ratios at 8:1, 64:1, and 512:1.We connect the data points of 64:1 and 512:1 in this setting using a straight line.

used three compression ratios (8:1, 64:1, and 512:1), and the prioritized coefficient strategy used five compression ratios (8:1, 64:1, 128:1, 256:1, and 512:1). So the total number of wavelet configurations tested was eighteen. Figure 10a and 10b show baseline renderings for these tests, and Figure 10c and 10d show two examples from the eighteen configurations. Visual inspection shows that data compression changes the streamlines in both structures compared to the baseline renderings

Figure 11 shows the NRMSE of the radial-enstrophy profile for the two structures after data compression. Both NRMSE plots show that the three wavelet settings using prioritized coefficients yield significantly lower errors than Haar+multi-resolution. In addition, when using prioritized coefficients, the two CDF kernels always perform better than the Haar kernel at compression ratios up to 256:1, and the CDF 9/7 kernel has the lowest errors at most configurations.

This result is consistent with our findings from the critical structure identification analysis.

2.6.3 Summary From Two Analyses. Table 1 summarizes our findings regarding how accuracy compares over wavelet configuration. In this table, the multi-resolution+Haar configuration serves as the baseline result, with the accuracy gains from other configurations shown comparatively. Only compression ratios of the form 8^N are shown, since those are the only ratios supported by the multi-resolution strategy. In all cases, the prioritized coefficients strategy using a CDF kernels perform best. The two CDF kernels perform similarly in most cases, although CDF 9/7 outperforms CDF 8/4 several times. Finally, the improvement is very significant at finer representations, but less noteworthy at very coarse representations.

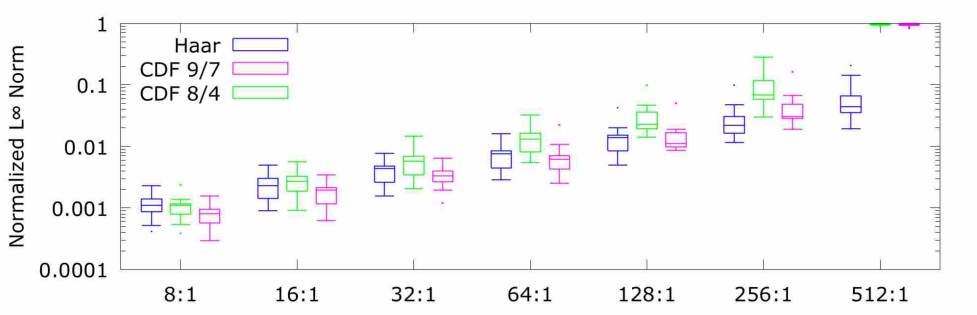
2.7 Further Evaluation of Accuracy, Storage Cost, and Execution Time

Our evaluation in the previous section helps illuminate tradeoffs between compression and accuracy in real-world scientific analyses. With this section, we report statistical error measurements, storage overhead incurred by the prioritized coefficient strategy, and execution time in three subsections, respectively.

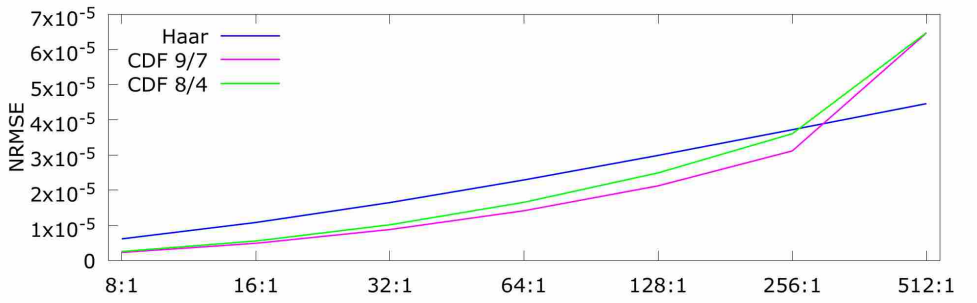
2.7.1 Statistical Error Measurements. While the focus of our study was on evaluating wavelet efficacy for established analyses, we also wanted to understand traditional statistical error measurements. The measurements we chose to perform were the L^∞-norm and the root mean square error. We chose these two metrics because they measure extreme differences and average differences, respectively. Specifically, the L^∞-norm captures the largest possible point-wise difference between the original and compressed data, and RMSE provides an average error across all vertices in the volume.

Table 1. Accuracy summary of different wavelet configurations. The multi-resolution+Haar configuration serves as the baseline, and the improvements from other configurations are shown comparatively.

Wavelet Configurations	Analysis 1 Proportion of		Analysis 2 NRMSE for	
	FN	FP	S1	S2
(8:1 Comp.)				
Multi-res+Haar	1x	1x	1x	1x
Prioritized+Haar	11.63x	9.16x	19.31x	3.74x
Prioritized+CDF 8/4	16.40x	13.60x	21.97x	9.93x
Prioritized+CDF 9/7	24.27x	16.35x	26.72x	21.88x
(64:1 Comp.)				
Multi-res+Haar	1x	1x	1x	1x
Prioritized+Haar	3.32x	3.59x	4.43x	3.86x
Prioritized+CDF 8/4	5.31x	3.07x	8.19x	14.05x
Prioritized+CDF 9/7	5.34x	3.80x	16.40x	14.05x
(512:1 Comp.)				
Multi-res+Haar	1x	1x	1x	1x
Prioritized+Haar	1.39x	0.83x	1.43x	2.21x
Prioritized+CDF 8/4	1.37x	1.39x	1.11x	1.30x
Prioritized+CDF 9/7	1.38x	1.38x	1.39x	1.27x



(a) Compression errors measured by Normalized L^∞-norm. Normalization is performed by scaling the absolute L^∞-norms by the maximum enstrophy in DS1. Each box in this box plot represents a distribution of the L^∞-norm over thirteen time slices of DS1. Note that the Y-axis is on a logarithmic scale.



(b) Compressor errors measured by normalized RMSE. Normalization is performed by scaling the absolute RMSE values by the maximum enstrophy in DS1. Each line in this chart represents an average of DS1's thirteen time slices.

Figure 12. Statistical measurements of wavelet compression errors for all time slices of DS1.

We performed our calculations by directly comparing every pair of corresponding vertices from the original and compressed data for all thirteen time slices of DS1. Because this comparison is meaningful only when the compressed data has the same mesh resolution as the original data, the multi-resolution strategy was not used. Figure 12a represents the normalized L^∞-norm using box plots, and Figure 12b shows the normalized RMSE (NRMSE). The CDF 9/7 kernel consistently performs the best up to 128:1 compression with the L^∞-norm, and up to 256:1 with RMSE. This result is consistent with our previous findings. We also note that while the CDF 8/4 kernel generally outperforms the Haar kernel in evaluations both in the previous section with real-world analyses and in the RMSE evaluation of this section, it yields larger L^∞-norm values than the Haar kernel.

2.7.2 Storage Overhead. Table 2 shows file sizes at different compression ratios for the first time slice of DS1 (256GB in raw form). The rest of the time slices have the same file size since they the same mesh resolution. We tested both multi-resolution and prioritized coefficient schemes. Because the prioritized coefficient scheme essentially introduces the same storage overhead regardless of wavelet kernel, we only report results from the Haar kernel. This table shows that the multi-resolution scheme achieves ratios close to ideal, i.e., the file sizes are very close to being in proportion with the compression ratio. It also shows that the prioritized coefficient scheme introduces approximately more than 50% overhead in storage.

The multi-resolution scheme is able to achieve full storage savings since the scheme can store its coefficients in the order generated by the forward wavelet transforms; the addressing of coefficients is implicit. The slight increases over the compression ratio are from meta data stored in the file.

Table 2. File sizes for wavelet-compressed data in GB. Our test data set was 256GB, and was compressed using the Haar kernel with multi-resolution and prioritized coefficient strategies. The actual achieved compression ratios are shown in parentheses. The four "N/A" entries are ratios that the multi-resolution scheme does not support.

Comp. Ratio	Ideal	Multi-resolution	Prioritized
1:1	256.0	256.0179 (0.99:1)	274.1094 (0.93:1)
8:1	32.0	32.0022 (7.99:1)	50.1094 (5.11:1)
16:1	16.0	N/A	25.0938 (10.20:1)
32:1	8.0	N/A	12.5781 (20.35:1)
64:1	4.0	4.0003 (63.99:1)	6.3125 (40.55:1)
128:1	2.0	N/A	3.1719 (80.71:1)
256:1	1.0	N/A	1.5938 (160.62:1)
512:1	0.5	0.5000 (511.97:1)	0.7969 (321.24:1)

The prioritized coefficient scheme must order coefficients based on their information content. This re-ordering requires tracking their addresses, and, in our study, the addressing is explicit. We note that, in the image processing space, encoders such as SPIHT (Kim & Pearlman, 1997) and SPECK (Pearlman, Islam, Nagaraj, & Said, 2004) are able to avoid this overhead with complex encoding strategies. However, their approaches require byte-scaling floating point data to integers, which may introduce additional information loss.

2.7.3 Execution Time. When used as an in situ compressor, wavelet compression introduces computational overhead during writing (to perform the forward wavelet transform that compresses the data) and reading (to perform the inverse wavelet transform that decompresses the data). In this section, we examine these computational overheads with different wavelet configurations, as the multi-resolution and prioritized coefficients compression strategies have different characteristics. For the multi-resolution strategy, the computational cost is closely related to the compression ratio. This is because more aggressive compression is achieved by applying the wavelet transform repeatedly to the data,

thus introducing more computational burden. In our study, we performed the wavelet transform three times (and thus achieved a compression ratio of 512:1). For the prioritized coefficient strategy, the computational cost is independent of the compression ratio, because we always reconstruct the meshes at their native resolutions.

We ran our tests on a subset of DS1 that measured 4GB in raw form. We did not use the whole data set since any given node of a supercomputer will be operating only on a portion of the overall data set.

Our experiment used one compute node on Maverick, a machine at the Texas Advanced Computing Center. Compute nodes on this machine have 20 CPU cores and 256GB system memory. We used community software (VAPOR) to perform the wavelet compression. This software was multi-threaded when using the prioritized coefficient strategy, i.e., it spawned 20 threads on our test machine. However, the software ran single-threaded when using the multi-resolution strategy.

Table 3 reports the run time to perform the forward Discrete Wavelet Transform (DWT) and the Inverse Discrete Wavelet Transform (IDWT), with each measurement averaged over ten runs. The results show that the prioritized coefficient strategy, even with improved parallelism, introduces significantly larger computational costs than the multi-resolution strategy. The CDF 9/7 kernel, which performed best in our accuracy evaluations, is the slowest to execute. Finally, we notice that the IDWT operations take significantly less time than DWT, meaning that it is much faster to decode wavelet-compressed data in a post hoc analysis.

While the run times to apply DWT are greater than ten seconds, they may be acceptable for in situ usage. We envision wavelet compression running only when the simulation wants to output data; since this happens infrequently and,

Table 3. Time cost, in seconds, to perform Discrete Wavelet Transform (DWT) and Inverse Discrete Wavelet Transform (IDWT) on a 4GB subset of DS1.

	Multi-resolution	Prioritized Coefficients		
	Haar	Haar	CDF 9/7	CDF 8/4
DWT	11.4297	12.9927	14.2177	13.9134
IDWT	5.2971	2.2621	3.8233	3.0584

since I/O is a costly operation, the overhead from the compression is likely small in comparison.

2.8 Other Experiments Informing Wavelet Efficacy

A few more experiments reported in other chapters of this dissertation also inform the efficacy of wavelets, though they are designed for other purposes. These experiments include:

- visualization of compression artifacts and statistical error measurements on a Lulesh (Karlin, Keasler, & Neely, 2013) simulation, in Section 3.4.3.
- statistical error measurements on multiple variables from three simulations (Ghost (Mininni, Alexakis, & Pouquet, 2006), Tornado (Orf, Wilhelmson, Lee, Finley, & Houston, 2017), and CloverLeaf3D (Mallinson et al., 2013)), in Section 5.4.
- visualization and error measurement of pathline integration over 220 time slices, in Section 5.5.1.
- visualization and error measurement of isosurface analysis, in Section 5.5.2.

Findings from these experiments together with the evaluations in this chapter provide a better understanding on the efficacy of wavelet compression for scientific simulation data.

2.9 Conclusion

We performed an evaluation study on the efficacy of wavelet configurations for turbulent-flow simulations. Our approach took two existing visual analytics routines and repeated them on compressed data sets, varying over compression strategies (multi-resolution and coefficient prioritization), wavelet kernels (Haar, CDF 9/7, and CDF 8/4), and compression ratios. We complemented this analysis with traditional statistical error measurements, additional information on storage requirements, and computational overhead for applying wavelet transforms. In total, this study informs the tradeoffs between accuracy, storage cost, and execution time when applying wavelets to turbulent-flow data.

Our findings show that the coefficient prioritization approach and the CDF kernels provide significant benefits over the multi-resolution schemes that rely on (tri)linear filtering to produce coarsened data (Haar as an example). Since the experiments we performed were diverse and their results were consistent, we believe that these findings are likely to generalize to other scientific visualization usages as well. While our findings match best practices from the image processing community, our focus on quantifying the accuracies achieved for domain scientist's analyses allow us to determine the magnitude of the benefit for real world settings. Interestingly, the variation in overall accuracy was quite high across analysis routines, emphasizing the importance of keeping the final usage in mind.

CHAPTER III

VIABILITY OF IN SITU WAVELET COMPRESSION

This chapter is mainly based on a collaborative publication (Li, Larsen, et al., 2017). Hank Childs and I finalized the experiments to run; I performed the experiments with valuable help from Matt Larsen; and all of us contributed to the analysis and understanding of the experiment results. Hank Childs and John Clyne also helped me edit the final paper to have a clearer presentation.

While we aim to fit wavelet compression in the in situ compression+post hoc analysis paradigm, its performance impacts to simulations are not fully understood, especially on today's large-scale supercomputers. This is because, although the computational costs are relatively predictable, the I/O performance is subject to many factors on supercomputers, and the parallel filesystems themselves tend to have less intuitive characteristics. In this chapter, we take a first step to understand these impacts: we test an in situ lossy wavelet compressor together with a proxy simulation on Cheyenne (CISL, 2017), the flagship supercomputer of National Center for Atmospheric Research. We run a weak scaling test with up to 1,000 compute nodes and measure the performance impacts in both computation and I/O. Based on data collected from this experiment, we discuss various in situ wavelet compression considerations, and our major contributions in this chapter are 1) analysis of a weak scaling experiment on a large-scale supercomputer, and 2) derivation of an empirical equation regarding I/O tradeoffs for in situ compression.

3.1 Related Work

3.1.1 In Situ Processing and Analysis. In situ processing has been in use since the 1960s, where it was used to print images of a running simulation directly to microfilm (Zajac, 1964). Later, in situ processing and analysis, also

referred to as "co-processing," has been used for computational steering that allows users to alter a simulations as it executes for a variety of reasons including preventing crashes, changing numerical solvers, and performance optimization (Heiland & Baker, 1998; Mulder, van Wijk, & van Liere, 1999). More recently, the growing I/O gap has reinvigorated research into in situ techniques, so analysis can take place while the data is still in memory or the amount of data is reduced before it is written to disk. For example, LibSim (Whitlock, Favre, & Meredith, 2011) and Catalyst (Fabian et al., 2011) facilitates in situ visualization with existing visualization packages.

There have also been many interesting ideas for operators that enable exploratory visualization with the in situ + post hoc strategy, including the following examples. Cinema's (Ahrens et al., 2014) main strategy is to transform the data to images, with the idea being that many, many images will still be smaller than simulation data, and that exploration can happen by loading successive images as if they were being generated by a traditional visualization program. The idea with Lagrangian basis flows (Agranovsky et al., 2014) is to transform vector field data into pathlines in situ, and then interpolate new pathlines post hoc from the extracted ones. This technique was shown to be more accurate than saving vector field data, and used less storage as well. As some final examples, Analysis-Driven Refinement (Nouanesengsy, Woodring, Patchett, Myers, & Ahrens, 2014), or ADR, prioritized the data to save based on the analyses that would be performed, while Lehmann et al. (Lehmann & Jung, 2014) explored a multi-resolution technique in both space and time. A comprehensive report by Bauer et al. (Bauer et al., 2016) surveys even more in situ reduction directions.

3.1.2 Compression of Scientific Data.

Compression has been well explored to reduce the size of scientific data. Lossless techniques (e.g., Fpzip (Lindstrom & Isenburg, 2006) and FPC (Burtscher & Ratanaworabhan, 2009)) retain the full integrity of data, but hardly achieve impressive reduction factors. As a result, more techniques opt for a lossy compression, with examples being ZFP (Lindstrom, 2014), SZ (Di & Cappello, 2016; D. Tao, Di, Chen, & Cappello, 2017), ISABELA (Lakshminarasimhan et al., 2011), and wavelet based compressors (Kim & Pearlman, 1997; Norton & Clyne, 2012; Tang, Pearlman, & Modestino, 2003). Lossy compression techniques can achieve aggressive reduction factors while still allowing meaningful analysis to be carried out, as reported in analyses of turbulent flow data (Li et al., 2015), climate data (Baker et al., 2014; Woodring et al., 2011), and physics-based proxy simulations (Laney, Langer, Weber, Lindstrom, & Wegener, 2013).

For in situ compression, the compressor's ability to utilize parallelism becomes critical. ISABELA divides data into windows and compresses each window independently (Lakshminarasimhan et al., 2011). VAPOR divides a volume into domains (typically of size 64^3) and applies wavelet compression on each domain independently (Norton & Clyne, 2012). Another piece of our work (Li, Marsaglia, et al., 2017), used in the remainder of this chapter and described later in detail in Chapter IV, contributed a wavelet compression implementation using data parallel primitives, achieving portable performance among multiple architectures. The work reported in this chapter differs from previous work in that we focused on the compressor's performance impacts to simulations in an in situ setting.

3.2 Experiment Overview

3.2.1 Software Used. Our experiments ran under the umbrella of ALPINE Ascent in situ infrastructure (Larsen et al., 2017), a production version of Strawman (Larsen et al., 2015). The infrastructure consists of two main parts: an interface to simulations and a hybrid-parallel library that provides a distributed-memory layer on top of shared-memory parallel algorithms in VTK-m (Moreland et al., 2016); together they enable fast prototyping of in situ algorithms.

On the simulation side, our study used the Lulesh (Karlin et al., 2013) proxy-application, a 3D Lagrangian shock hydrodynamics code. Lulesh implements the Sedov test problem, which deposits initial energy at one corner of a cube, and propagates a shock wave from the origin outward to the rest of the cube. In terms of computation, Lulesh uses thread-level parallelism (via OpenMP) for calculation within a domain, and process-level parallelism (via MPI) for multiple domains.

For wavelet compression, we used an implementation introduced in (Li, Marsaglia, et al., 2017) that we integrated into ALPINE's hybrid-parallel library. This implementation fits well in our in situ experiment settings because it has comparable performance with the best multi-thread CPU wavelet compressors, and is versatile enough to enable high-performance compression on GPUs. It implements filter-bank based wavelet transforms, and we used the CDF 9/7 (Cohen et al., 1992) wavelet kernel in this study. For lossy compression, it uses the coefficient prioritization approach, which keeps the wavelet coefficients that contain the most information (i.e., largest magnitudes) and discards the rest. For example, a 64:1 compression means keeping $1/64th$ of all coefficients. We note that the CDF 9/7 + coefficient prioritization combination was determined to be the most suitable wavelet configurations for compression uses in Chapter II.

In terms of computation, this implementation achieves thread-level parallelism using data parallel primitives on single compute nodes, but does not have process-level parallelism between compute nodes. After finishing compressing, each node independently writes its data in the compressed form to disk via an `fwrite()` function call of the C programming language.

3.2.2 Simulation with In Situ Compression. Our simulation and compression code runs in an in situ setting: at the end of each simulation cycle, simulation variables are passed to the compression code to perform compression, and then written to disk in the compressed form. Multiple variables are processed one at a time, during which the simulation is suspended. As a result, each complete cycle consists of one simulation step and multiple compression and writing steps (for multiple variables). Specifically, our experiment has seven variables: pressure, energy, relative volume, artificial viscosity, and the x, y, z components of velocity. One might think compressing multiple variables together could better exploit the available parallelism. However, our in situ wavelet compressor is already a parallel implementation, so simultaneous compression would not make a big difference in computational time. (It would make more difference in memory consumption though, since multiple variables need to stay in memory simultaneously.) Finally, while data being saved so frequently does not reflect real world usage, it is sufficient for our study.

Our in situ compression is "tightly coupled." This means, each compute node not only performs simulation on a domain, but also compresses the data of the same domain without data movement through the network.

Disk I/O is handled individually by each compute node as well. After processing a variable of a domain, the compute node writes its content to disk as

an individual file no matter whether there is compression or not, until all variables are processed.

We note that we use MPI barriers in the code to keep all MPI ranks in sync, meaning that they are always in the same stage of performing simulation, compression, or writing. We feel this represents real-world usage in terms of bursty I/O coming from all nodes at the same time.

Finally, our compression code treats the compression ratio 1:1 as a special case — it does not perform any compression on the data, and directly passes the data to the file writer. In our tests, this configuration acts as the baseline case to compare and measure in situ compression impacts.

3.2.3 Experiment Runs. We ran our experiments on Cheyenne, the flagship supercomputer of National Center for Atmospheric Research. Each compute node is equipped with two 18-core Xeon CPUs at 2.3GHz (36 cores in total) and 64GB memory. Both simulation and compression software were compiled with GCC-6.3.0, and they used 64-bit floating point values to carry out their computation.

With the Lulesh simulation, we fixed the size of each MPI rank's domain size to be 320^3 (320^3 cells and 321^3 vertices). The entire simulation then scales up by using more domains/MPI ranks, making it a weak scaling problem. Lulesh supports the number of MPI ranks being cubics of natural numbers, i.e., 1^3, 2^3, 3^3, etc.

We vary two parameters for experiment runs:

- number of MPI ranks: 1, 8, 27, 64, 125, 216, 343, 512, 729, and 1,000;
- compression ratio: 1:1, 16:1, 64:1, and 128:1;

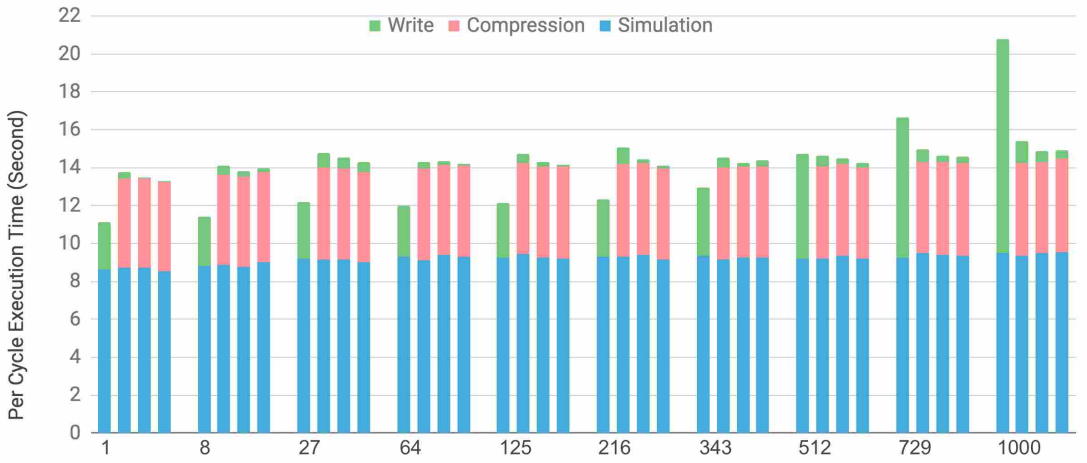


Figure 13. Per cycle execution time breakdown of 40 experiment runs. Each group of four uses the same number of compute nodes (X labels), but different compression ratios: 1:1, 16:1, 64:1, and 128:1 (from left to right within each group).

The number of MPI ranks is essentially the number of compute nodes we use, since we assign one MPI rank to a node. The total number of tests we performed is then $10 \times 4 = 40$.

3.3 Results

3.3.1 Execution Time Impacts. We present the execution times in Figure 13. Each column is one experiment run. These columns are grouped into groups of four; each group has experiment runs with the same number of compute nodes. The four columns within each group differ in their compression ratios; they are 1:1, 16:1, 64:1, and 128:1 from left to right. Note the 1:1 column has no compression time.

Each column provides an execution time per cycle averaged from five cycles. Though each cycle has one simulation step, it has seven compression and writing steps for seven variables. (see Section 3.2.2). The sum of the seven compression and writing times are reported here. The runs were performed during a lull on the

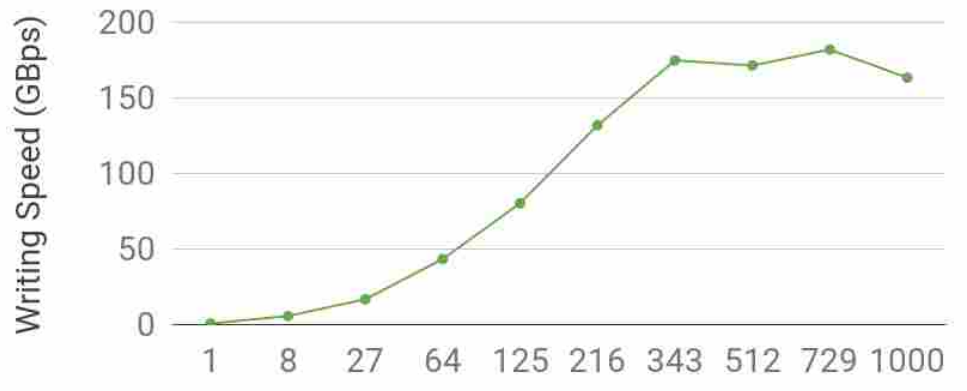


Figure 14. Achieved parallel filesystem writing speed with different number of compute nodes (X-axis).

machine in the middle of the night, in an effort to minimize I/O contention with other jobs running on the supercomputer.

Experiment results show that simulation time stays relatively steady at around 9 seconds per cycle, no matter how many nodes are in use and whether or not in situ compression is involved. Wavelet compression adds computational overhead to every cycle. This overhead is mostly consistent as well (around 5 seconds per cycle), because the wavelet transform and coefficient thresholding steps are both independent of the final compression ratio.

3.3.2 I/O Performance Analysis. The I/O time shows more interesting results than computational time considering the huge difference in the amount of data being written to the filesystem. With our experiment settings, each node generates a fixed amount of raw data per cycle. The total amount of data is then proportional to the number of nodes. However, looking at the writing time of raw data (left-most columns of each group of four), they are by no means proportional to the total amount of data being written.

We consider this interesting I/O behavior to be attributed to the very large aggregate bandwidth of parallel filesystems. This bandwidth is large enough that a small set of compute nodes are not using up all of it — the I/O is constrained by the network between individual nodes and the parallel filesystem. After the number of concurrently-writing nodes grows past a certain point, the I/O bottleneck starts to shift to the parallel filesystem, when its bandwidth gets saturated. Looking at our results, the raw data writing time is almost constant up to 216 nodes, and then starts to grow with the number of nodes.

We compared our results with the specs of the test system (Cheyenne of NCAR), which is equipped with a 200GBps filesystem. The amount of raw each compute node generates each cycle is roughly 1.84GB. The achieved aggregate I/O is then calculated using the total amount of data and writing time, which is presented in Figure 14. It tops out at approximately 170GBps starting from 343 nodes, which is plausible given the hardware specs. We achieved this I/O rate partially because we run our experiments when the supercomputer is relatively idle, and in-production simulations are likely to face more contentious conditions.

The I/O time with compression is much less consistent, meaning that writing time for the same-ratio compressed data vary considerably from one run to another. Compared to raw data writing time, it is almost always taking a larger percentage of time than its data size fraction. We consider this to be due to the relatively large latency of parallel filesystems, which introduces a lot of variance to writes with small amount of data. The only certainty for such scenarios is that writing compressed data takes much less time than raw data.

3.4 Viability of In Situ Compression

3.4.1 Overall I/O Viability. With in situ compression, computational overhead is incurred while the actual I/O is most likely reduced. We thus consider the *overall I/O*, which includes both computational and the actual I/O cost. In our experiment, computational overhead counts for the majority of overall I/O, which may or may not improve over writing raw data.

In the case where the number of compute nodes is small, the achievable aggregate I/O is able to grow as the number of compute node grows. The computation of the compression then needs to be performed fast enough to improve the overall I/O, at least faster than writing raw data to disk. This can be considered as a performance lower bound. The in situ wavelet compression implementation we tested fails in this criterion, and it caused overall I/O to grow with less than 512 nodes (Figure 13).

In the case with a large number of compute nodes, their concurrent I/O requests may top out past the aggregate I/O of the parallel filesystem. In situ compression is then possible to improve overall I/O here. Given a filesystem with aggregate I/O bandwidth V_{aggr}, N compute nodes each generating D_{domain} amount of data, the I/O time to write the raw data is

$$T_{raw} = \frac{N \cdot D_{domain}}{V_{aggr}}.$$

Assume an in situ compressor that takes T_{comp} time in calculation and achieves a compression ratio of R, the overall I/O for the same N compute nodes is then

$$T_{in-situ} = T_{comp} + \frac{N \cdot D_{domain} r}{V_{aggr} \cdot R}.$$

The overall I/O $T_{in-situ}$ improves over T_{raw} when the compression time satisfies the following equation:

$$T_{comp} < \frac{N \cdot D_{domain} \cdot (R-1)}{V_{aggr} \cdot R} \ .$$

With lossy compression, compression ratios are usually big enough that $(R-1)/R$ can be approximated to be 1. The equation can then be re-written as:

$$N > \frac{T_{comp} \cdot V_{aggr}}{D_{domain}} \ .$$

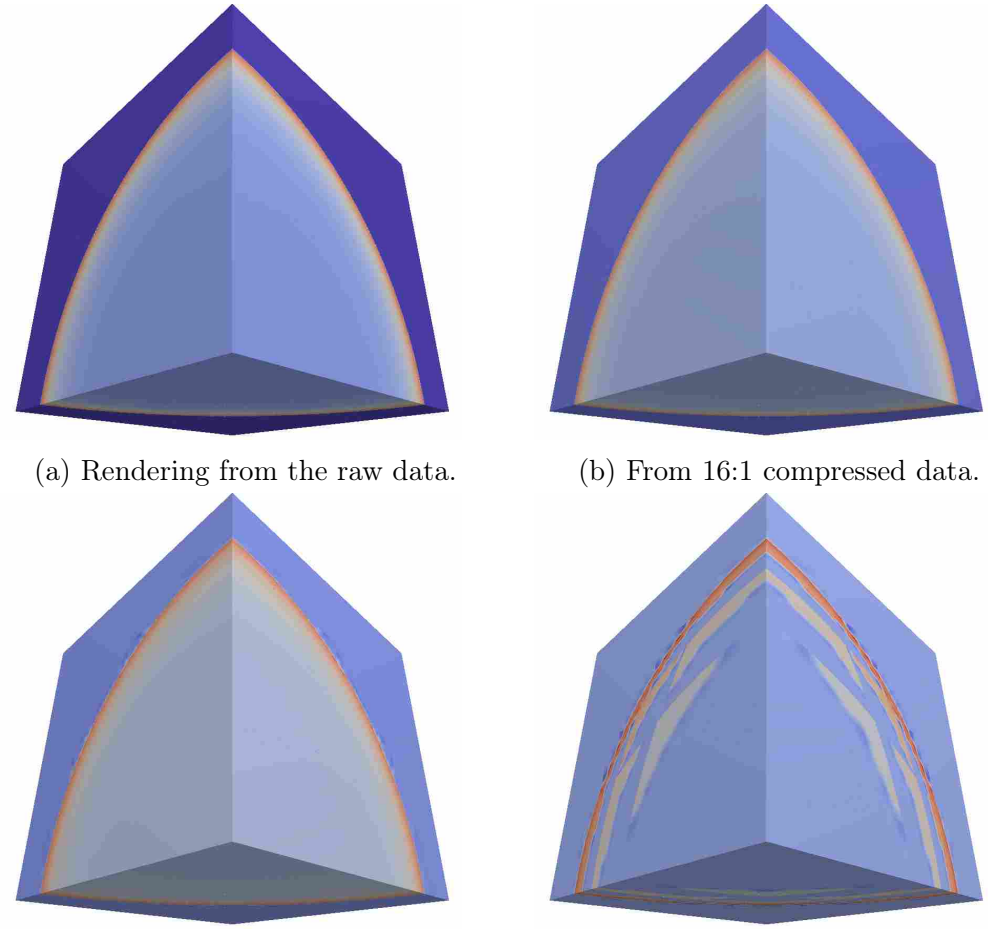
This equation means that with enough compute nodes, in situ compression can always improve the overall I/O. First, without much surprise, this threshold is proportional to the computational overhead T_{comp}, so faster compression lowers this threshold. Second, this threshold is also proportional to the rate between aggregate filesystem bandwidth and the domain size (V_{aggr}/D_{domain}). Given that the domain size may be constrained by the memory capacity of compute nodes, and that the memory capacity growth keeps outpacing the bandwidth growth, this trend further lowers this threshold. In our experiment, 512 was estimated to be the threshold so the overall I/O was improved with 729 and 1,000 nodes; the I/O time of 1,000 nodes was almost cut in half. We anticipate simulation runs with larger numbers of nodes would benefit even more.

Besides the factors we have analyzed here, the overall I/O of real-world simulation runs is also subject to other variations, including hardware specifications, parallel filesystem characteristics, and even other users concurrently using the system. An accurate analysis of the overall I/O is even more challenging with these variations. We believe the trend is a more important takeaway, which is that larger simulations are more likely to benefit from a reduced overall I/O,

and the amount of benefit is going to grow as the relative I/O bandwidth keeps decreasing.

3.4.2 Memory Viability. Memory overhead is another consideration of in situ compression, since many simulations are already bound by available system memory. The wavelet compressor we used does not work in a streaming fashion. Instead, it requires an extra buffer space for each variable it processes. That means, for a variable of size S, it introduces a memory overhead of $2S$. A simulation usually consists of multiple variables. In the case where the in situ compressor fully makes use of the available parallelism, this memory overhead is not multiplied since these variables can be processed one by one. The $2S$ memory overhead thus becomes independent of the total number of variables, making in situ compression more viable. In the case where the compressor is not fully parallelized, multiple variables need to be processed simultaneously to exploit the available parallelism. The $2S$ memory overhead is then multiplied, making in situ compression less viable. The wavelet compressor in our experiment falls into the first case, so it introduced memory overhead that is twice the largest variable at 321^3 resolution, or 529MB in total, which was acceptable for our test supercomputer system.

3.4.3 Data Integrity Viability. Data integrity is a universal concern of all lossy compression techniques, but the acceptance of information loss greatly varies. This acceptance is usually determined by factors such as the nature of the intended analysis, the accuracy requirement of a particular application, as well as resource limitations such as I/O and storage. Given that this paper focuses on performance impacts of in situ compression, we present a visualization and simple statistics of the resulting compression but choose not to go into detailed analysis.



(a) Rendering from the raw data.

(b) From 16:1 compressed data.

(c) From 64:1 compressed data.

(d) From 128:1 compressed data.

Figure 15. Ray tracing renderings of the pressure field from Lulesh: a shock wave propagates in a cube. They are from the raw data, and compressed data with ratios of 16:1, 64:1, 128:1. Note that most artifacts emerge on the shock wave front.

Table 4. Error measurements from compression for each compression ratio (first column from left). Both root-mean-square error (RMSE) and L-∞ norm values (second column) are normalized by the range of data. The rest columns are evaluations of five data fields: energy (e), relative volume (v), pressure (p), artificial viscosity (q), and x-velocity (xd).

		e	v	p	q	xd
16	RMSE	$6.7e-8$	$4.1e-8$	$3.5e-4$	$2.2e-4$	$1.1e-3$
	$L-\infty$	$9.2e-7$	$4.7e-7$	$5.3e-3$	$4.4e-3$	$1.5e-2$
64	RMSE	$7.7e-7$	$4.8e-7$	$4.9e-3$	$5.8e-3$	$1.0e-2$
	$L-\infty$	$1.8e-5$	$1.6e-5$	$1.0e-1$	$1.2e-1$	$2.0e-1$
128	RMSE	$1.7e-6$	$1.1e-6$	$1.0e-2$	$1.2e-2$	$1.8e-2$
	$L-\infty$	$3.8e-5$	$2.3e-5$	$2.6e-1$	$2.9e-1$	$3.4e-1$

Figure 15 presents ray tracing results of the pressure field of this shock wave at time step 4,300, in a 128^3 cube. While the 16:1 result looks almost indistinguishable from the baseline, the 64:1 result has clear "ripples" along the shock wave front, since that is where most of the energy resides. The 128:1 result is even more deteriorated with not only rough front, but also artifacts in the volume. Table 4 presents error measurements for multiple data fields at the same time step. The normalized root-mean-square error provides an average deviation, and the normalized $L - \infty$ norm provides the maximum point-wise difference. We used a 128^3 cube for this visualization instead of sizes in our performance tests (320^3) because the per-voxel differences are still prominent in smaller volumes; renderings look more similar to each other in a 320^3 cube. Again, data integrity is highly analysis-dependent and a complex topic by itself. Interested readers should review recent publications such as (Laney et al., 2013; Li et al., 2015; Woodring et al., 2011).

3.4.4 Storage Viability. In situ compression reduces the storage cost regardless of its performance: either the same data takes less storage, or the same storage is able to hold more data. If data integrity requirements are satisfied, both scenarios are welcome for scientists. Thus, the storage viability is really a benefit rather than a concern.

3.5 Conclusion

We studied performance impacts of in situ wavelet compression on a scientific simulation. Based on results and analysis of a weak scaling experiment on a large-scale supercomputer, we gained better understanding of the I/O impacts of in situ compression, and derived an empirical equation to quantify when we can expect in situ compression to improve the overall I/O. Finally, we argued

that in situ compression is a viable alternative to post hoc and in situ analysis by discussing various aspects of its viability.

CHAPTER IV

WAVELETS FOR EMERGING ARCHITECTURES: MANY-CORE ARCHITECTURES

This chapter is mainly based on a collaborative publication (Li, Marsaglia, et al., 2017). Hank Childs, Christopher Sewell, and I produced the original plan for this study. While I performed the major software development in this study, Christopher Sewell provided valuable guidance in adopting the novel programming paradigm (data parallel primitives with VTK-m), and John Clyne greatly helped me understand the many details of wavelet compression algorithms. Nicole Marsaglia and Vincent Chen contributed a native GPU implementation for performance comparison in this study (Section 4.5.2). Hank Childs, Christopher Sewell and John Clyne helped me analyze and understand the experiment results, as well as edit the final submission.

In terms of my research goals of this dissertation, this chapter addresses the third one, i.e., a focus on a specific kind of emerging hardware. In particular, this chapter addresses many-core architectures. The next chapter (Chapter V) still addresses the third research goal, but will focus on another kind of emerging hardware: burst buffers.

4.1 Motivation

The computational capacity of HPC systems never stops growing. To develop even faster HPC systems, there are two approaches. The first is to make individual processing units run faster. In most cases, this is a synonym for higher clock frequencies. The second approach is to use more processing units. Traditionally, this is a synonym for more compute nodes, or CPU cores. Since the 2000s, it has become increasingly difficult to achieve significant advances with the

first approach. The major limiting factor has been the amount of power needed to drive very high clock frequencies (often above 3GHz), and the associated heat dissipation requirements at those frequencies. As a result, the second approach has been playing a more important role to achieve better performance. For example, the number of compute nodes grows from one generation of HPC systems to the next, and these compute nodes tend to contain more cores through the use of multi-core CPUs.

More recently, the "using more processing units" approach is also prevailing in the designs of processors that are intended to use within a compute node. More specifically, manufacturers are in favor of packing more cores into a processor rather than improving a single core's performance. At times, the performance of a single core may even decrease. Examples of such processors include the Intel Xeon Phi, which contains 60 to 70 cores with each running at 1.1 to 1.4GHz, and the Nvidia Tesla, which contains thousands of cores with each running at hundreds MegaHertz. Processors with this massive parallelism usually have very distinct characteristics, and architectures of this type are often referred to as "many-core" architectures.

The emergence of many-core architectures imposes two challenges to application developers: the first being how to harness the amount of parallelism available, and the second being how to program with *portable performance*. Here portable performance means that the same program performs well not only on one many-core architecture, but also on others, not to mention the existing multi-core architectures of today. In some cases, portable performance is a premise of making use of the massive parallelism, since with a growing number of architectures thriving simultaneously, developers can no longer afford to design and tune programs targeting a specific architecture. In fact, algorithms and programs with

portable performance are expected to play larger roles in the future of HPC, and portability is becoming a key factor deciding how applicable an application could be in various usages.

This chapter investigates the portability of wavelet compression. Specifically, it focuses on designing a wavelet compressor that will be hardware-agnostic and yet still achieves high performance on each architecture it runs on. Ideally, this design can "future-proof" our code to run not just on today's architectures, but also tomorrow's. Recent research has demonstrated that designing code using *data parallel primitives* (DPPs) as building blocks is a promising direction for achieving this goal. Therefore, the research involved with this work — and the contribution of this chapter — is to re-think wavelet compression using data parallel primitives and to demonstrate the efficacy of the resulting algorithm on multiple architectures.

4.2 Related Work

4.2.1 Parallel Wavelet Transforms on CPUs. Domain decomposition is a popular yet effective approach for achieving parallel processing on CPUs. Using this approach, an entire domain is decomposed into smaller subdomains and each subdomain is processed individually. For 2D matrices, the JPEG 2000 image compression standard employs this approach (Acharya & Tsai, 2004). A similar application on multi-node settings is also reported in (Uhl, 1995). For 3D volumes, VAPOR (Clyne et al., 2007; Clyne & Rast, 2005), an open-source visualization package with a wavelet compression component, decomposes incoming volumes into 64^3 cubes by default and then processes them in parallel. Although domain decomposition has the advantage of simplicity, the technique can suffer from blocking effects along subdomain boundaries, which arise from wavelet artifacts on finite-length input boundaries.

More complicated parallel approaches treat the entire domain as a whole while performing wavelet transforms in parallel. These approaches eliminate blocking effects, but introduce inter-process communications. Nielsen et al. (Nielsen & Hegland, 2000) developed a parallelization strategy that eliminates a time-consuming distributed matrix transpose, and demonstrates strong scalability. Chaver et al. (Chaver, Prieto, Pinuel, & Tirado, 2002) partitioned 2D matrices into stripes and studied the performance differences between X-partitioning and Y-partitioning. Chadha et al. (Chadha, Cuhadar, & Card, 2002) further developed a partitioning strategy where intermediate information exchanges are restricted to neighboring processors. Though proven to be effective on multi-core CPUs and distributed systems, it is unclear how similar strategies would perform on many-core architectures. Also, these strategies seem to have, for the most part, not considered 3D volumes.

4.2.2 Parallel Wavelet Transforms on GPUs. Parallel wavelet transforms on GPUs have been predominantly conducted within the CUDA framework (Nickolls, Buck, Garland, & Skadron, 2008). Natural parallelization strategies on GPU include row-based and column-based processing, which uses one GPU thread to process a row or a column of an image at a time (Ao, Mitra, & Nutter, 2014; Enfedaque, Auli-Llinas, & Moure, 2015). Domain decomposition is also used to get the CPU and GPU to work together: a CPU sends subdomains to a GPU to process, and retrieves back the results one-by-one (Franco, Bernabé, Fernández, & Ujaldón, 2010).

A trend in GPU-based wavelet transforms is to exploit the many memory hierarchies on GPU devices to achieve higher speedups, including discussions on the use of shared memory (Franco et al., 2010), texture memory (Garcia & Shen, 2005),

and even registers (Enfedaque et al., 2015). While these fine-grained tunings are very effective in making the most out of the hardware, they usually require a good amount of GPU programming skills, and the performance gains are not guaranteed to translate to another version of hardware.

Finally, we point out that an important use of GPU wavelet transform is to perform on-demand decompression at rendering time. The idea is to postpone decompression to the latest possible stage of the rendering pipeline, which is on GPUs, to reduce the expensive data movement costs. An example of this use is GST (Krajcevski, Pratapa, & Manocha, 2016), where "supercompressed" textures are decoded on GPUs. A detailed survey on this topic is also available at (Balsa Rodríguez et al., 2014).

4.2.3 Visualization Algorithms With DPPs. Several studies have investigated how to re-think a specific algorithm in the framework of data parallel primitives. They include Maynard et al. with thresholding (Maynard, Moreland, Atyachit, Geveci, & Ma, 2013), Larsen et al. with ray-tracing (Larsen, Meredith, Navrátil, & Childs, 2015) and unstructured volume rendering (Larsen, Labasan, Navrátil, Meredith, & Childs, 2015), Schroots and Ma with cell-projected volume rendering (Schroots & Ma, 2015), Lessley et al. with external facelist calculation (Lessley, Binyahib, Maynard, & Childs, 2016), Lo et al. with isosurface generation (Lo, Sewell, & Ahrens, 2012), and Carr et al. with contour tree computation (Carr, Weber, Sewell, & Ahrens, 2016). Our own work differs in that we are considering a different algorithm (wavelet transform).

4.2.4 Other State-of-the-art Floating Point Compressors. Motivated by the I/O bottleneck on supercomputers, several schemes are designed to specifically compress the floating-point data arising from numerical simulations.

Some representatives include Fpzip (Lindstrom & Isenburg, 2006), ZFP (Lindstrom, 2014), SZ (Di & Cappello, 2016), and ISABELA (Lakshminarasimhan et al., 2011). However, the ability of these schemes to perform well on multiple architectures is still not clear, and this work focuses on how to obtain portable performance for wavelet compression.

4.3 Data Parallel Primitives

In the data parallel paradigm, algorithms are made by composing together so-called data parallel primitives, or DPPs. A DPP specifies the pattern of how an input array is processed in parallel to produce outputs, while users take the responsibility to specify operations applied on each individual element. This user-specified operation is sometimes referred to as "functors" or "worklets." A benefit of using data parallel primitives is that execution details such as thread and memory management are abstracted away from general users, which in turn allows specific implementations to optimize for underlying architectures. Algorithm designers then re-think their algorithms using a relatively small set of data parallel primitives to harness the massive parallelism in modern architectures. Here we briefly describe a few data parallel primitives for demonstration purposes. Readers can consult work by Blelloch (Blelloch, 1990) for theoretical foundations and Nvidia's Thrust (Bell & Hoberock, 2011) for examples in an actual product.

Map is a simple yet powerful data parallel primitive — it maps each data element from the input array to an element in the output array. The input and output arrays thus have the same size. Map resembles a traditional `for` loop if there are no loop-carried dependencies. Elements are thus processed in parallel with arbitrary order.

Scan also maps an input array to an output array with the same size, but resembles a `for` loop that does have loop-carried dependencies. Scan can be efficiently executed in parallel in a bottom-up fashion.

Reduce uses all elements from the input array to produce a single output value, for example the sum or the maximum of the input array. Reduce can also be efficiently executed in parallel in a bottom-up fashion.

Scatter and **Gather** are data parallel primitives to facilitate data movement — individual elements are moved in parallel to or from designated locations assuming there are no index conflicts.

In practice, more complex data parallel primitives can be constructed by composing the basic data parallel primitives. This process is useful for providing fundamental algorithms, and an example of this is the **Sort** algorithm in Thrust.

4.4 Algorithm Description

Our compression algorithm consists of two primary steps: wavelet transformation followed by coefficient prioritization. The basics of wavelet transformation is already covered in Section 2.2, so here we detail a specific approach (filter-banks) we used for implementation in Section 4.4.1. We then briefly discuss coefficient prioritization with DPPs in Section 4.4.2, before we discuss more implementation specifics in Section 4.4.3.

4.4.1 Filter-bank Based Wavelet Transforms.

There are multiple approaches available to perform the wavelet transform, with filter banks (Strang & Nguyen, 1996) and lifting schemes (Sweldens, 1996) being most popular. We adopted the filter bank approach in this study because of its flexibility; different wavelets can be handled using different filter banks without dramatical changes to the program.

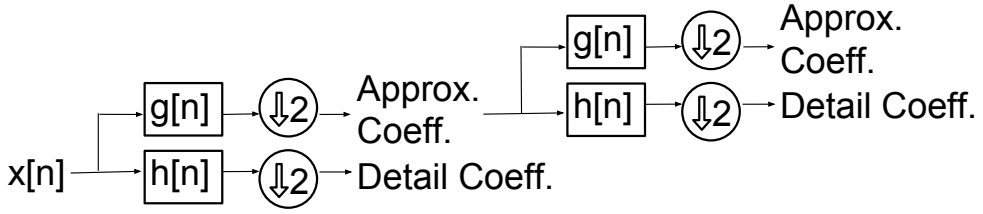


Figure 16. Illustration of a filter-bank based wavelet transform workflow. The input signal ($x[n]$) passes through a low-pass and high-pass filter ($g[n]$ and $h[n]$, respectively), and is then down-sampled by a factor of two, resulting in approximation and detail wavelet coefficients. This process is repeated on the approximation coefficients to create a second level wavelet transform.

With the filter bank approach, the core operation to calculate wavelet coefficients is discrete convolution. More specifically, we use a two-channel filter bank to perform wavelet transforms, with each filter convolving with the input array (signal) to produce wavelet coefficients. The first channel is a low-pass filter, and the resulting "approximation" coefficients provide a coarsened representation of the signal. The second channel is a high-pass filter, and the resulting "detail" coefficients contain the missing information from the low-pass filtering. The total number of output coefficients is doubled by convolving with two filters. A down-sampling step with a factor of two then restores the same number of coefficients to the input array. Despite downsampling, it is still possible to retain all information according to the Nyquist's rule: half of the frequency components passed through a filter, thus only half of the coefficients were needed to represent them (Nyquist, 1928).

The approximation coefficients are recursively transformed in the same manner — iterating through the filter banks until a stopping criterion is reached. This practice further decorrelates the approximation coefficients to achieve a better compression. Figure 16 illustrates a two-level wavelet transform workflow.

In practice, discrete convolution requires special care on the boundaries for finite-length input data. In the general case that the data is not periodic, the data array needs to be extended by half the filter length on both ends, so discrete convolution can perform as usual on the real data. Usually, the extension past the boundary uses the last few elements of the input data array. With an appropriate choice of convolution filter pairs, and careful boundary extensions, mathematically perfect reconstruction is possible with the number of wavelet coefficients matching the number of original samples.

Also from a practical standpoint, the down-sampling step in Figure 16 leaves opportunities to eliminate unnecessary calculation of coefficients, i.e., to skip calculation of coefficients that are meant to be discarded. This is achieved by performing discrete convolution with the low-pass filter on even indexed elements, and the high-pass filter on odd indexed elements.

4.4.2 Coefficient Prioritization. The second step of wavelet compression is to prioritize all coefficients and keep only the ones with the most information content. The heart of this process is a "sort" routine based on the magnitudes of the coefficients. After sorting, a decision is made (typically as input to the compression process) about how many coefficients to save. These coefficients are the largest values. The remaining coefficients are not saved, and treated as zeros during data reconstruction.

4.4.3 Implementation Specifics. We implemented our algorithm within the VTK-m framework (Moreland et al., 2016). VTK-m provides a set of platform-agnostic data parallel primitives to algorithm developers, which sits on top of architecture-specific back ends. Currently, VTK-m has two optimized back ends for its DPPs: CUDA (Nickolls et al., 2008) for Nvidia architectures, and Intel

TBB (Pheatt, 2008) for Intel architectures. Also, because of the high-level nature of VTK-m, some architectural specifics, such as the different kinds of memories on an Nvidia GPU, are not exposed to its users.

With regards to memory organization, our implementation keeps data in a row-major one-dimensional array regardless of its logical dimensionality. This design means we must face less-than-ideal memory access patterns when accessing data in columns or frames. One potential work-around is to transpose the matrix (or volume) to the desired orientation before performing wavelet transforms along that axis. However, in-place transposition for a matrix (or volume) with different sizes along each dimension is not trivial by itself. We did not choose this optimization for simplicity.

Our implementation supports four wavelets: three members from the CDF (Cohen et al., 1992) wavelet family (CDF 9/7, CDF 8/4, and CDF 5/3), and the Haar wavelet. We used the CDF 9/7 wavelets in this study because it is arguably the best for lossy compression usage.

Finally, we note that our implementation is already merged into the open-source VTK-m repository.

4.4.3.1 Wavelet Transform with DPPs.

We used the "Gather" data parallel primitive to perform signal extension. Gather naturally fits in here since it retrieves elements from designated locations of the signal to extensions (just like gathering). We use specific worklets to guide Gather to correctly handle different dimensionalities and extension directions (e.g., left, right, etc.). Though extending a signal is computationally light because of the small sizes of extensions, implementing them using a data parallel primitive has the additional benefit of avoiding potential data transfers between different computing environments (e.g.,

Algorithm 1 Worklet for 3D Wavelet Transform in the X Axis

Require: $signal$, $leftExt$, $rightExt$, $workIdx$ ▷ $workIdx$ is assigned by VTK-m
Ensure: $coefficients$

```
1: procedure TransformX( signal, leftExt, rightExt, workIdx, coefficients)
2:     (x, y, z) ← GetLogicalIndex( workIdx )
3:     if x is even then                          ▷ Because this is transform along X
4:         array ← ComposeX( signal, leftExt, rightExt, x, y, z )
5:         sum ← DiscreteConvolution( arrary, lowWaveletFilter )
6:         outIdx ← GetOutputIndexApproximationCoeff(x, y, z)
7:         coefficients[ outIdx ] ← sum
8:     else
9:         array ← ComposeX( signal, leftExt, rightExt, x, y, z )
10:        sum ← DiscreteConvolution( array, highWaveletFilter )
11:        outIdx ← GetOutputIndexDetailCoeff( x, y, z )
12:        coefficients[ outIdx ] ← sum
13:    end if
14: end procedure
```

between the host and a GPU). This is because DPPs can usually be scheduled to run on designated devices, which allows us to schedule them in the environment where data resides.

Wavelet transforms are carried out using a "Map" data parallel primitive. Details of the transforms, such as wavelet banks and convolution operations, are passed in as worklets. We implemented individual worklets for wavelet transforms in each dimensionality and direction; each worklet results in a slightly different Map that performs wavelet transform for one particular case. This practice reduces execution branches inside a worklet, which helps maximize the GPU performance. Algorithm 1 outlines a worklet performing 3D wavelet transforms along the X axis. It assumes that each row of the three-dimensional input is properly extended with an extension on both left and right side (`leftExt` and `rightExt`, respectively), and receives its own work index (`workIdx`) from the VTK-m scheduler, so each instance of the worklet performs convolution on one index: (x, y, z).

4.4.3.2 Coefficient Prioritization with DPPs.

For coefficient prioritization, we used the "Sort" data parallel primitive provided by VTK-m. VTK-m exposes platform-optimized sort when possible. Specifically, it exposes the parallel merge sort from Thrust (Bell & Hoberock, 2011) on GPUs, and the parallel quick sort from TBB on CPUs.

4.5 Study Overview

4.5.1 Experiment Overview.

We performed our experiments in two rounds. The first round focused on evaluating our own algorithm, while the second round focused on comparing with hardware-specific implementations.

4.5.1.1 Round 1: Evaluation of the VTK-m Approach.

This round was designed to better understand the basic performance of wavelet compression across multiple platforms. It varied two factors:

- Hardware architecture: multi-core CPU and GPU.
- Data sizes: 256^3, 512^3, $1,024^3$, and $2,048^3$.

We tested all data sizes on CPU, but skipped the $2,048^3$ data size on GPU due to the GPU memory capacity limitation. We also tested 1D and 2D data inputs for evaluation purposes, and their results yielded similar patterns to 3D inputs. Since 3D data sets are most relevant to HPC applications including simulations and scientific visualizations, we only report 3D results here. We report results from artificial data sets with Gaussian distributions, although the actual data values do not impact performance significantly, because the number of floating point operations and function invocations remains constant for each test size.

4.5.1.2 Round 2: Comparison with Platform Specific Implementations.

This round compared the VTK-m implementation with platform specific implementations for multi-core CPUs and CUDA GPUs, namely

VAPOR (Clyne et al., 2007; Clyne & Rast, 2005) for multi-core CPUs, and a native CUDA implementation for GPUs. These implementations represent the best practices on respective architectures, so they are good comparators for the VTK-m implementation. The total number of configurations for this round is four: VTK-m and VAPOR on multi-core CPUs, and VTK-m and CUDA on GPUs. Again, we opt to only report 3D test results as representatives, and each test is run with multiple problem sizes.

4.5.2 Software Specifications. There are three software packages used in our study: our VTK-m implementation, VAPOR, and a native CUDA implementation. Details about the VTK-m implementation are in Section 4.4.3, so this section focuses on VAPOR and the CUDA implementation.

VAPOR is an open-source software framework consisting of multiple components, including a GUI for post hoc exploration of wavelet-compressed data. For this study, we made use of the standalone wavelet compression utilities included with VAPOR. This program achieves parallel processing through domain decomposition, i.e., a large volume would be decomposed to fixed-sized blocks, and multiple blocks are processed individually and simultaneously using `pthreads`. Coefficient prioritization (described in Section 4.4.2) is performed individually within each block as well using the C++ STL sort.

The native CUDA implementation was written for our study. It followed implementation decisions discussed in (Scivoletto & Romano, 2016) with adaptations to our GPU. For example, we maxed out the number of threads per block on our GPU to be $1,024$ for larger throughput. Wavelet transforms in each direction (X, Y, and Z) are implemented as separate CUDA kernels for parallel processing. Data is always organized as one-dimensional arrays in the global

memory on the GPU without explicit use of shared memory. Thrust sort was used here during coefficient prioritization. Overall, this CUDA implementation has a very similar structure to its VTK-m counterpart, minus the platform-agnostic infrastructure from VTK-m.

Both CPU softwares (VTK-m+TBB and VAPOR) are compiled using GCC, and both GPU softwares (VTK-m+CUDA and native CUDA implementation) are compiled using NVCC with GCC. We turned on `-O2` optimization for all compilations.

4.5.3 Hardware Specifications. To support the tests described in Section 4.5.1, we used the following test systems; both systems are used in both rounds of our testing.

- **CPU System**: Dual socket Intel Xeon Haswell CPUs running at 3.2GHz. There are 16 cores in total, and each core is hyper-threaded to have 2 threads.
- **GPU System**: Nvidia Tesla K40 GPU. There are 2,880 cores in total, each running at 745MHz. This GPU also has 12GB on-board high speed memory.

4.6 Results

The results are organized following the two rounds of our experiments: Section 4.6.1 analyzes the performance of our algorithm over multiple architectures, and Section 4.6.2 compares our performance to hardware-specific implementations.

4.6.1 Performance Analysis of the Algorithm. We separately analyze multi-core CPU performance (4.6.1.1) and GPU performance (4.6.1.2).

4.6.1.1 Multi-core CPU Performance Analysis. Our first set of experiments studied strong scaling of the VTK-m implementation. We ran a baseline of a single core, and then ran additional tests with sixteen cores. In both cases, the compressed volume was the same size. Table 5 shows timing values

Table 5. Strong scaling study of VTK-m on 16 Xeon CPU cores. For each problem size, computation time is reported for both transform (shortened as XForm) and sort subroutines in seconds. The achieved speedup is reported in the last column.

Size	Subroutine	1-core	16-core	Speedups
256^3	XForm	4.72	0.33	14.30X
	Sort	1.36	0.22	6.18X
512^3	XForm	37.22	2.06	18.07X
	Sort	12.23	1.41	8.67X
$1,024^3$	XForm	298.67	16.22	18.41X
	Sort	103.75	13.32	7.79X
$2,048^3$	XForm	2512.10	131.40	19.12X
	Sort	884.53	93.18	9.49X

Table 6. Factor of computational time increase from a smaller to a bigger problem size. Values in this table are derived from the 16-core results in Table 5.

Size Incr.	XForm Time Incr.	Sort Time Incr.
$256^3 \rightarrow 512^3$	6.24X	6.41X
$512^3 \rightarrow 1,024^3$	7.87X	9.45X
$1,024^3 \rightarrow 2,048^3$	8.10X	7.00X

and speedup factors on four problem sizes. The results show that the transform subroutine achieves near perfect speedups (around $16X$), indicating that the worklet based approach is able to harness the additional CPU cores. In some cases, the speedup numbers are even higher than $16X$. We speculate this is due to the hyper-threading nature of the Xeon CPUs, since VTK-m sees 32 logical cores through TBB and launches 32 threads for computation. However, the sort subroutine only has speedups from $6.18X$ to $9.49X$. This reduced performance is expected, since sorting requires coordination between the cores.

Our second set of experiments looked at the execution time increase as the problem size grows. We calculate the ratio of execution times using the sixteen core results and list them in Table 6. The problem size grows by 8 at each step. This table shows that both transform and sort subroutines take close to $8X$ more

Table 7. Wavelet transform and sorting time on a Tesla K40 GPU in seconds. The factor of time increase from the previous problem size is indicated in parentheses.

Size	XForm Time	Sort Time
256^3	0.0463	0.0445
512^3	0.3177 (6.86X)	0.3834 (8.62X)
$1,024^3$	2.4419 (7.69X)	3.1766 (8.29X)

Table 8. Theoretical and achieved occupancy of our wavelet compressor on a Tesla K40 GPU. The transform subroutine was implemented as a worklet, and the sort subroutine was a data parallel primitive provided by VTK-m.

	Theoretical Occupancy	Achieved Occupancy
XForm	75%	70.3%
Sort	50%	49.4%

time to finish processing the next problem size. This result indicates that this implementation is not slowing down as we approach data sizes up to $2,048^3$.

4.6.1.2 GPU Performance Analysis. Our first set of experiments measure raw performance on the GPU. Table 7 provides the time the GPU takes to perform wavelet compression on three data sizes: 256^3, 512^3, and $1,024^3$. We did not test the $2,048^3$ data size because it exceeded the memory capacity on our GPU. These tests show a significant performance boost compared to 16-core CPUs. Given that this is the same code base compiled on two very distinct architectures, it shows that the performance can be portable. Also, the execution time increase is in line with the problem size growth: it takes roughly $8X$ more time to solve an $8X$ larger problem.

Secondly we use *occupancy* reported by the Nvidia Visual Profiler to assess the efficiency of the VTK-m program. In Nvidia's model, adjacent threads are grouped into warps. There is a maximum number of warps that can be concurrently active on a Streaming Multiprocessor depending on the underlying hardware. Occupancy is then defined as the ratio of active warps to the maximum

number of active warps supported by the Streaming Multiprocessor. It is not always possible to achieve 100% occupancy for a general program because of limiting factors in compilation and GPU invocation specifics (more details can be found in Nvidia documentation (Nvidia, 2015)). As a result, the Nvidia Visual Profiler reports a theoretical occupancy as well as an achieved occupancy. The achieved occupancy cannot reach the theoretical occupancy when the scheduler is not able to issue sufficient instructions because of data or instruction dependencies. We report both occupancy metrics in Table 8 for two major subroutines in our algorithm: wavelet transform and sort.

The occupancy results are generally good, with the wavelet transform worklet achieving a higher occupancy. This is because of the nature of the wavelet transform that worklets working on individual convolutions are more independent with each other than sorting. For both subroutines, the Nvidia Visual Profiler suggests that the occupancy is large enough that further improvements in occupancy may not improve performance.

We note that for large-scale simulations on supercomputers, a $1,024^3$ cube is on a par with problem sizes a single compute node normally processes. We argue that the achieved compression speed on GPUs, e.g., under six seconds for a $1,024^3$ cube, is likely fast enough to fit within in situ requirements and facilitate the in situ+post hoc strategy to alleviate I/O constraints.

4.6.2 Comparisons With Hardware-Specific Software.

4.6.2.1 VAPOR. As previously discussed, VAPOR achieves parallel processing via domain decomposition and `pthreads` (see Section 4.5.2). For the tests on different size data sets, we maintained the number of total subdomains at 64, allowing VAPOR to make full use of the multi-core CPU. In our case, the test

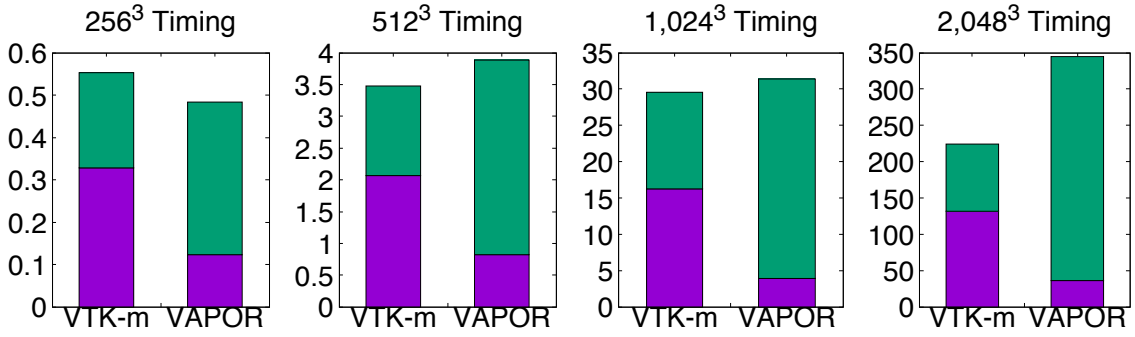


Figure 17. Comparison of execution time (in seconds) between VTK-m and VAPOR. The purple region is for wavelet transforms, and green is for sorting.

machine has 16 physical cores which are hyperthreaded to appear as 32 cores, so VAPOR launches 32 threads with each one processing two subdomains. VAPOR processes each subdomain following the transform and sort subroutines as the VTK-m implementation does. We note that the local sort within each subdomain actually results in fewer calculations than the global sort in VTK-m, but for simplicity in comparison, we consider the sort time to be local for VAPOR and global for VTK-m.

Figure 17 shows the performance comparison between VTK-m and VAPOR. These results show that VTK-m and VAPOR have comparable performance with VTK-m being faster in three of the four test sizes. However, a more prominent difference is how they allocate time differently between their two subroutines. While VTK-m spends more than half its time performing wavelet transforms, VAPOR spends less than a quarter, especially as the problem size grows. This result is interesting since it shows that our DPP-based wavelet transform is $3X$ to $4X$ slower than the best implementations on CPU.

We speculate two design choices by VAPOR contributed to its superior performance: slice-by-slice data processing, and transposition for cache alignment. Both design choices aim to better use the caching mechanism on CPUs. First, a

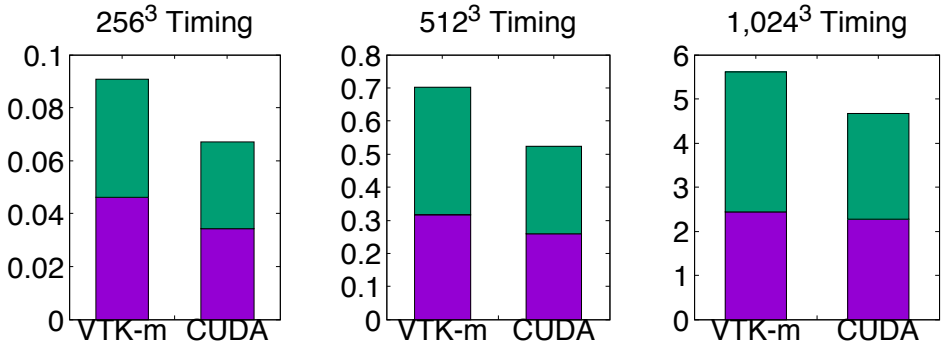


Figure 18. Comparison of execution time (in seconds) between VTK-m and CUDA. The purple region is for wavelet transforms, and green is for sorting.

slice from the subdomains that VAPOR processes is most likely to fit into the last level of cache in modern CPUs. For example, a slice from 512^3 subdomains is 1MB in 32-bit `float` or 2MB in 64-bit `double` type, which can easily fit into the 20MB L3 data cache per CPU socket (40MB in total) in our test system. Second, VAPOR transposes data to align arrays in storage to the one dimensional wavelet transforms about to be performed, further increasing cache utilizations in smaller but faster L2 and L1 caches. On the contrary, our data parallel primitive based transform schedules worklets to process arrays as long as one entire volume dimension without certain orderings, hardly making good use of the caching mechanism.

In terms of the time cost for sorting, the STL sort employed by VAPOR does not perform as well as VTK-m's sort, which is TBB's sort for CPUs. One might think that replacing the STL sort in VAPOR to TBB sort could be a simple solution to increase VAPOR's performance. However, it would not be that easy, since VAPOR is already parallelizing across cores for the domain decomposition, and thus the sort for each subdomain can only use a single thread.

4.6.2.2 Native CUDA Implementation.

Figure 18 compares the performance difference between the VTK-m and native CUDA implementations.

Since they share similar parallelization strategies (see Section 4.5.2), this comparison actually quantifies the performance overhead of VTK-m on GPUs. As the results show, this overhead is always within 40% of the CUDA performance. In fact, this overhead has a trend to decrease as data size grows (i.e., from 35% at 256^3 to 20% at $1,024^3$).

4.7 Conclusions and Future work

This chapter explored a new approach to implement a wavelet compression algorithm, distinguished in its aim to achieve portable performance over multiple architectures. This new approach made use of the data parallel primitive paradigm, which aims to future-proof for emerging architectures. We showed that our performance is comparable with two hardware-specific softwares on multi-core CPUs and Nvidia GPUs. The GPU comparison also quantifies the VTK-m overhead to be no more than 40% of its native CUDA counterpart.

For future work, we would like to explore techniques that enable us to process larger data sets on GPUs despite their constrained memory capacity, for example, the greatly enhanced unified memory from CUDA 8.

CHAPTER V

WAVELETS FOR EMERGING ARCHITECTURES: BURST BUFFERS

This chapter is mainly based on a collaborative publication (Li, Sane, et al., 2017). Hank Childs and John Clyne helped me to form the game plan for this study. Leigh Orf and Pablo Mininni provided data from their simulations, as well as analyses which are used repeatedly in this study. I performed most of the experiments except a pathline integration implementation (Section 5.5.1), which was contributed by Sudhanshu Sane. Hank Childs and John Clyne were also heavily involved in experiment result analysis as well as paper editing.

In terms of my research goals of this dissertation, this chapter addresses the third one with a focus on another type of emerging hardware: burst buffers. The previous chapter (Chapter IV) addressed the same research goal with a different focusing hardware: many-core architectures.

5.1 Introduction

5.1.1 Motivation. Modern computers are equipped with a hierarchy of memory. Usually this hierarchy consists of registers and a few levels of caches on the same processor dies of CPUs, system memory for the multiple processing units in the same compute node, and the parallel filesystem for the entire HPC system. Tape-based storage would contribute one more level of hierarchy to systems that are equipped with it. Overall, this hierarchy of memory aims to ensure the most frequently used data is accessible with the least amount of time.

With the increasing I/O bottleneck which we keep referring to in this dissertation, the discrepancy between two levels in this hierarchy, main memory and the parallel filesystem, is growing rapidly. Following the same spirit of our existing memory hierarchy, modern architectures could have one more level of hierarchy

in between, mitigating this discrepancy. In fact, with the development of one particular technology, solid state drive (SSDs), this extra hierarchy level is emerging in more and more systems, and is often referred to as "burst buffers." These burst buffers usually provide a higher throughput than the main filesystem, although with a smaller capacity. For example, the Wrangler system from The Texas Advanced Computing Center has an SSD-based storage achieving an aggregate rate of 1TB/s, which is 40 times higher than the primary filesystem. Its capacity, on the other hand, is about $1/20^{th}$ of the primary storage.

How to make use of this burst buffers and an extra memory hierarchy is a hot research topic. Its superior capability to handle intermittent large amount of I/O requests motivates us to re-think our in situ compression workflow, especially temporal compression opportunities between time slices.

To date, in a typical in situ compression workflow, a compression operator is inserted as the simulation saves its state, such that the operator is applied to single time slices one by one. This compression can be quite effective, since neighboring data points in a mesh are often coherent (very smooth) and many compression operators perform best on coherent data. Simulation data is often temporally coherent as well, and thus our research looks at compression across time as well as space. This alternate approach, referred to as "spatiotemporal compression" in this chapter, enables compression operators to exploit temporal coherency while continuing to take advantage of spatial coherency.

Spatiotemporal approaches were not feasible previously on supercomputers, since simulation codes have traditionally been memory-constrained and thus only had room to operate on a single time slice. But burst buffers create opportunities to temporarily store multiple time slices as a "window" and apply spatiotemporal

compression on this window. With this chapter, we investigate how feasible and effective spatiotemporal compression is on scientific data.

5.1.2 Spatiotemporal Compression Propositions. This chapter explores the benefit of including the time dimension for wavelet-based compression. While other techniques besides wavelets could be considered, we find wavelets to be an excellent operator for our evaluation, since the reference point of spatial-only compression is so well studied. Moreover, the feasibility of wavelet-based spatiotemporal compression was previously impractical due to the relatively wide temporal "window size" required. Our results show that spatiotemporal (4D) wavelet compression is superior to spatial (3D) wavelet compression for each of the following propositions:

- **P1:** Improve the accuracy, while maintaining temporal resolution and storage costs.
- **P2:** Reduce storage costs, while maintaining temporal resolution and accuracy.
- **P3:** Increase temporal resolution, while maintaining storage costs and accuracy.

Our study consists of two phases. First, we evaluate the efficacy of the approach with respect to our three propositions, as well as the performance impacts of operating on multiple time slices jointly. This phase also includes the study of multiple available parameters with particular relevance to wavelet transforms in the time domain, and helps inform their best combinations in practice. Second, we consider real-world visualization use cases which demonstrate how spatiotemporal compression improves analyses by providing more information per byte.

5.2 Related Work

5.2.1 Spatiotemporal Wavelet Compression. The benefits of wavelet-based, spatiotemporal compression are not well explored for scientific data, especially compared to studies considering only the spatial domain. Some of the earliest work on spatiotemporal wavelets was performed by Villasenor et al. (Villasenor, Ergas, & Donoho, 1996) and Trott et al. (Trott, Moorhead, & McGinley, 1996), who both applied a one-dimensional wavelet filter bank over all four dimensions on seismic reflection and fluid dynamics data, respectively. Zeng et al. (Zeng, Jansen, Unser, & Hunziker, 2001; Zeng, Jansen, Marsch, Unser, & Hunziker, 2002) established the feasibility of spatiotemporal wavelet compression of time-varying echocardiography images. In their earlier work (Zeng et al., 2001), the authors pointed out that the degree of coherence present may differ between dimensions, and thus warranted different handling. Lalgudi et al. (Lalgudi, Bilgin, Marcellin, & Nadar, 2005; Lalgudi, Bilgin, Marcellin, Tabesh, et al., 2005) evaluated 4D spatiotemporal compression on functional MRI (fMRI) data obtained as a time series of 3D images of the brain. Wang et al. (C. Wang, Gao, Li, & Shen, 2005) employed multi-resolution representations from 4D wavelet transforms in their visualization framework for time-varying data.

In all of the preceding work the authors found significant, albeit varying, benefit to 4D spatiotemporal compression over 3D spatial compression. Our work differs in the following ways: 1) our application domain is floating point data arising from numerical simulation; 2) we evaluate information loss with respect to key visualization algorithms, including algorithms that are sensitive to cumulative errors over time; 3) we evaluate the impact of various parameters of wavelet transforms in the context of temporal compression; and 4) we provide an evaluation

of how 4D compression works with Solid State Drives (SSDs) now found on HPC systems. All these differences target simulation data on HPC systems, which is a less studied space.

5.2.2 Other Temporal Compression Techniques. Motion compensated prediction (MCP) is a family of techniques stemming from video compression, such as the MPEG standard (Bosi et al., 1997). Researchers have explored the use of MCP on time-varying scientific data, for example, in (Guthe & Straßer, 2001; Ibarria, Lindstrom, Rossignac, & Szymczak, 2003; Sanchez, Nasiopoulos, & Abugharbieh, 2008). In theory, MCP could be useful for Eulerian type flow computations, since the fields move through the grid rather than the grid following the flow. However, it is not well understood how MCP's premise that pixels are moving in groups affect its application on scientific data.

Recent visualization research has also looked at spatiotemporal compression techniques specifically designed for scientific data. Ibarria et al. (Ibarria et al., 2003) proposed a "Lorenzo predictor" that operates on arbitrary dimensions, and a compression scheme based on it. Lakshminarasimhan et al. (Lakshminarasimhan et al., 2011) introduced ISABELA as a lossy spatiotemporal compression technique. ISABELA sorts data sequences in a window before performing B-Spline fitting to reduce fitting errors. Lehmann et al. (Lehmann & Jung, 2014) extended ISABELA by using a snapping mechanism to further improve the compression rate in certain cases. Finally, Agranovsky et al. (Agranovsky et al., 2014) employed a Lagrangian flow-based approach for vector field data that results in reduced file sizes and increased accuracy over spatiotemporal intervals.

5.2.3 Burst Buffers. The idea of burst buffers has been proposed to cope with the exploding data pressure from scientific applications. Scientific

applications typically have well defined execution phases. For example, they alternate between computation and I/O phases, which results in bursty and non-overlapping I/O. This is one of the fundamental motivations for designing burst buffers. The majority of research on burst buffer usage is recent. Liu et al. (Liu et al., 2012) designed a simulator of the burst buffer for the IBM Blue Gene/P architecture. Bing et al. (Xie et al., 2012) characterized output burst absorption on Jaguar and furthered quantitative models of storage system performance behaviors. BurstMem is a prototype burst buffer system developed by Teng Wang et al. (T. Wang et al., 2014). It is built on top of Memcached which is an open source, distributed caching system. BurstMem enhances Memcached by using a log-structured data organization with AVL indexing for fast I/O absorption and low-latency, semantic-rich data retrieval, coordinated data shuffling for efficient data flushing, and CCI-based communication for high-speed data transfer. BurstMem is able to speed up I/O performance of scientific applications by up to $8.5X$ on leading supercomputers. In total, these works demonstrate that the burst buffer is a viable element for spatiotemporal compression, especially since they are being incorporated into many newer supercomputers.

5.3 Method

5.3.1 Processing in Windows. Incorporating spatiotemporal compression into a simulation's output process requires careful consideration of memory. The length of the time dimension can vary widely from tens to thousands depending on the application. For simulations that may already be memory constrained, even with the availability of aforementioned, emerging deep memory hierarchies, retaining large numbers of time steps in memory may be challenging-to-impossible, thus limiting possibilities for temporal wavelet transform.

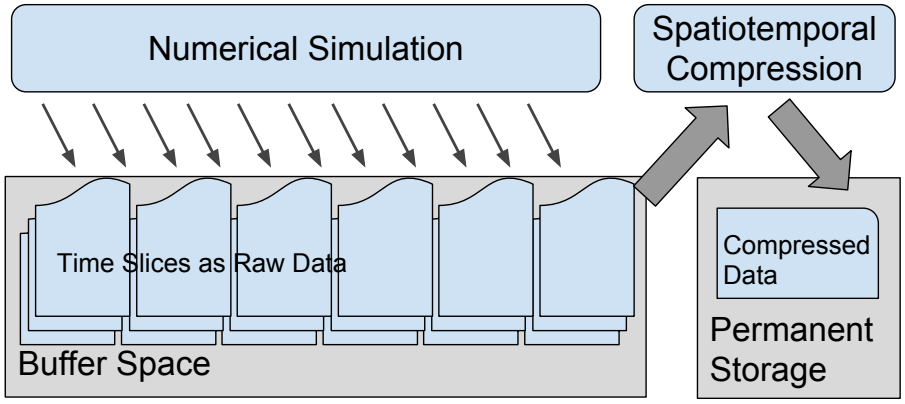


Figure 19. Our spatiotemporal compression workflow with a buffer space. For a window size of T, a simulation code writes T raw time slices to a buffer space. Then, for each variable, a compressor reads in the variable from the T time slices and applies spatiotemporal compression. The resulting compressed data is written to permanent storage. The process then continues for the next temporal window.

To address this issue in our implementation, we partition all time slices into "windows" and apply spatiotemporal compression on each window independently. Figure 19 illustrates our workflow.

At its peak, a buffer space will contain $T \times S \times N$ bytes, given a window size T, a single variable of S bytes, and N variables computed by the numerical simulation. We note that S reflects the number of grid points in this equation. Since each of the N variables can be compressed one at a time, the required available system memory is smaller, specifically $T \times S$. We believe many simulations can spare this much memory provided T is small, since simulations often allocate buffers for temporary usage, and these buffers can be used for our compression.

Wavelet transforms within each window involve two steps: first spatial and second temporal. In the first step, the spatial transform is essentially the same as what we described in Section 2.2.2. Here we specifically perform the "non-standard decomposition" (Burrus et al., 1998; Stollnitz et al., 1996) along three spatial dimensions: a single pass of the one-dimensional forward wavelet transform

is applied first along the X axis, then Y, and then finally Z. The filter bank is recursively applied up to J times, where allowable values of J are determined by properties of the filter bank and signal length described later. The output of this step is wavelet coefficients after spatial decorrelation.

In the second step, coefficients from the first step are first partitioned temporally into chunks of fixed size that we term the *window size*. We then apply a one-dimensional wavelet transform in time at each grid point location for each window. For a N^3 grid, N^3 temporal wavelet transforms are applied per window. We note that the relatively smaller window size, compared to the lengths of spatial domains, limits the levels of wavelet transform that can be applied.

After a time window is spatially and temporally transformed, a third step then takes place to compress the coefficients. Given a target compression ratio $n : 1, (n > 1)$, we find the coefficient with the $(num\_of\_coefficients)/n$ largest magnitude, and discard coefficients with magnitudes smaller than the threshold (i.e., they are treated as zeros in our study) to achieve a $n : 1$ compression ratio.

5.3.2 Temporal Domain Considerations. We consider two spatiotemporal compression parameters that require extra care: the aforementioned window size and wavelet kernel. The choice of window size is connected to the wavelet kernel choice; they need to be reconsidered under the premise that the window size is limited by the number of time slices that can fit in computer memory.

Larger window sizes allow more levels of wavelet transform to be performed before boundary conditions dominate the calculation. In turn, more levels of wavelet transform are favorable because they can exploit coherence at multiple scales. In our implementation, for a given window size, we set J, the number of

levels of wavelet transform, using the following equation:

$$J = \lfloor \log_2 \frac{window\_size}{filter\_size} \rfloor + 1, \tag{5.1}$$

where filter size is determined by the wavelet kernel. The two parameters, window size and wavelet kernel, interact in such a way that a smaller window size limits the available levels of wavelet transform to perform, while a wavelet kernel with a smaller filter size can potentially increase the level of wavelet transforms possible.

In terms of wavelet kernel choices, the Cohen-Daubechies-Feauveau (CDF) wavelet kernel (Cohen et al., 1992) has proven to be a suitable choice for scientific data compression as well as virtually all compression applications involving aperiodic data such as image compression (Li et al., 2015; Taubman, 2000; Unser & Blu, 2003; Woodring et al., 2011). The CDF family of *biorthogonal* wavelets are the only compactly supported wavelets besides the Haar kernel that preserve symmetry across scales, and thus allow for a non-expansive transform of finite signals when the signal boundaries are similarly symmetrically extended (Usevitch, 2001). A potential difficulty with the CDF 9/7, however, is that its relatively wide filter size (nine) may limit the levels of transforms that are practical with small window sizes. For example, with a window size of ten, CDF 9/7 is only able to do one level of transform. Another kernel from the same wavelet family and having satisfactory compression performance, the CDF 5/3 kernel, has a filter size five, and thus permits two levels of transform. The best practice is thus not trivial. Section 5.4.2.1 presents evaluation results on different combinations of wavelet kernels and windows sizes.

5.4 Our Study

This section explores spatiotemporal wavelet compression, aiming to identify best practices regarding various parameters, as well as to quantify benefits gained

from exploiting temporal coherency. The baseline results we use for comparisons are from 3D wavelet compression with CDF 9/7 wavelet kernels, which was found to be a top choice for compression in Chapter II. CDF 9/7 is also what we used for the spatial step of our spatiotemporal wavelet transform (see Section 5.3.1). Section 5.4.1 describes the experiments, Section 5.4.2 describes the results, and Section 5.4.3 describes performance impacts. Section 5.4.4 relates these results back to our three propositions for domain scientists. Finally, Section 5.4.5 discusses the limitations we observed regarding spatiotemporal compression.

5.4.1 Overview of Experiment Parameters. We varied five parameters to study:

- Wavelet kernel: "CDF 9/7" and "CDF 5/3";
- Window size in the time domain: 10, 20, and 40;
- Data set and variable: 7 variables from 3 simulations;
- Temporal resolution: 3 or 4 options for each variable;
- Compression ratio: 4 or 5 steps from 8:1 to 128:1.

5.4.1.1 Wavelet Kernel and Window Size. Wavelet kernel and window size together determine how many levels of the wavelet transform can be performed in the time dimension, which has a direct impact on the compression result (see Section 5.3.2). We consider two wavelet kernel candidates on the temporal domain: CDF 9/7 and CDF 5/3 with filter sizes nine and five, respectively. Though CDF 9/7 yields better compression results in most settings, the narrower width of CDF 5/3 is also compelling. We also consider three window sizes: 10, 20, and 40. With these window sizes, CDF 9/7 is able to perform 1, 2, and 3 levels of wavelet transform, respectively, and CDF 5/3 is able to perform one more level at each window size (i.e., 2, 3, and 4 levels) due to a shorter filter size.

5.4.1.2 *Temporal Resolution.*

Compression in the time domain relies on the data coherency between available time slices, which is a direct result of available temporal resolution. A higher temporal resolution provides more data coherency, and thus is more suitable for temporal compression. This experiment quantifies how temporal resolution affects spatiotemporal compression accuracy.

We focus on results from a forced incompressible hydrodynamic turbulent flow simulation from the Ghost (Mininni et al., 2006) simulation code to study temporal resolution in this subsection. More details of the physics of a similar Ghost simulation are given by Mininni et al. (Mininni, Alexakis, & Pouquet, 2008). We ran the simulation and saved time slices at a base temporal resolution: every 100^{th} simulation cycle. We denote this base resolution as "1." When experimenting with various temporal resolutions, we reduce this base resolution by using every 200^{th} cycle and using every 400^{th} cycle. We denote the lowered temporal resolution as "1/2" and "1/4," respectively.

These chosen temporal resolutions are required for certain analyses on the finer structures in a turbulent flow. For example, the sampling frequency needs to be high enough that a complete rotation of an eddy is captured by approximately 4 samples, otherwise the eddy may deform substantially. Smaller scale eddies take less time to make a complete rotation, so they require the simulation code to save time slices more frequently to enable a meaningful study.

5.4.1.3 *Data Sets and Variables.*

In addition to the Ghost data set, which simulates a homogeneous turbulent flow, we also used two other simulation outputs in our evaluation: Tornado and CloverLeaf3D. Tornado (Orf et al., 2017; Orf, Wilhelmson, & Wicker, 2016) simulates the dynamics of an F5 tornado, and CloverLeaf3D (Mallinson et al., 2013) solves the compressible Euler equations in

hydrodynamics settings. We focused our study on temporal regions of interest for the simulations being carried out. In the case of Ghost and Tornado, we used only the later portion of the simulation when interesting phenomena occur for both, which actually imposes greater challenge for compression. For CloverLeaf3D, we used the entire life span of this simulation.

We used two or three different data fields from each of these simulations. For the CloverLeaf3D data sets, we used energy and the X-component of velocity, since these are important, dynamic variables with distinct physical characteristics. For the Ghost and Tornado data sets, we used enstrophy and again the X-component of velocity, for the same reasons. We also used the cloud ratio scalar for the Tornado data set, since this variable determines what the clouds look like to human eyes. In terms of grid size, the two fields from Ghost are on 512^3 grids; the three fields from Tornado are on $490^2 \times 280$ grids with 280 in the Z direction; and CloverLeaf3D X-velocity has a size of 97^3 while CloverLeaf3D energy has a size of 96^3, since the latter is a cell-centered field. We note that the tornado domain analyzed in this paper is significantly smaller than the full model domain, and yet is also typical of what tornado researchers would use. This is because for studies of tornado morphology, features dozens of kilometers away from the tornado are not of primary interest.

We ran our experiments on these variables using multiple temporal resolutions with the same notations described in Section 5.4.1.2. The base resolution (res=1) differs from simulation to simulation though: every simulation cycle in CloverLeaf3D; every one simulation second in Tornado; and every 100^{th} simulation cycle in Ghost. We note that the twice coarser resolution (res=1/2) for

the Tornado data is essentially the common practice of our collaborating scientist, who normally uses every two simulation seconds in his research.

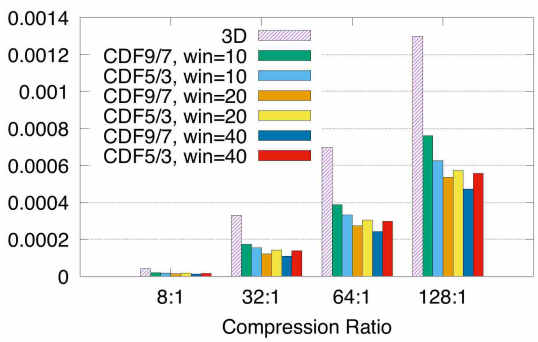
5.4.1.4 Compression Ratios. Our implementation of wavelet compression works by retaining a portion of the wavelet coefficients and discarding the rest. We refer to the parameter that determines the size of the retained portion as compression ratio. With our notation, 8:1 means one eighth of the total coefficients are retained, and so on.

When compressing multiple time slices, the process of discarding coefficients happens on each time slice individually with spatial compression, and on the entire group of time slices with spatiotemporal compression. Given a certain compression ratio, the total number of retained coefficients stays the same no matter spatial or spatiotemporal compression. We use compression ratios 8:1, 16:1, 32:1, 64:1, and 128:1 in most of the following tests.

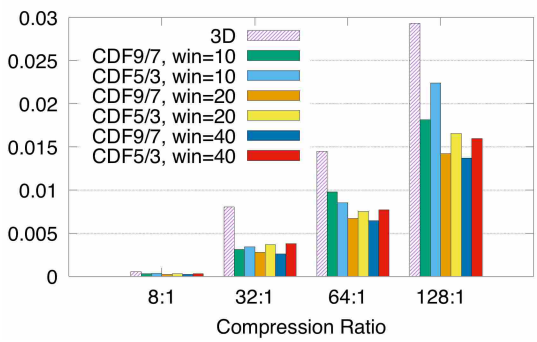
5.4.2 Results. The results considered in this section are error measurements from comparing a wavelet-compressed data set (either 3D or 4D) with its original version in a point-wise fashion. We use two statistical metrics: normalized root mean square error (NRMSE) and normalized L^∞ norm. NRMSE provides an average error across all vertices in the volume, while the normalized L^∞ norm captures the largest deviation introduced to a single data point.

To understand the effects of our study parameters, we ran experiments in three phases, varying some factors and holding the rest constant for each phase. The study results are reported in the following three subsections, and will be revisited as we consider the three propositions for domain scientists in Section 5.4.4.

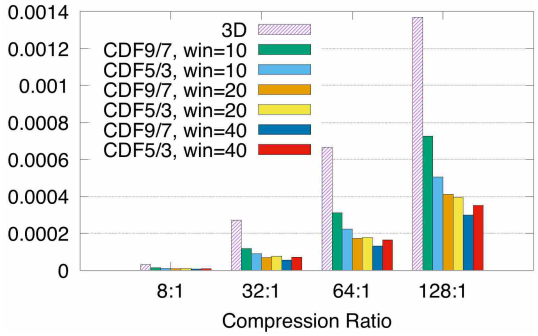
5.4.2.1 Wavelet Kernel and Window Size. This phase looks at the relationship between wavelet kernels and window sizes, with a goal of finding



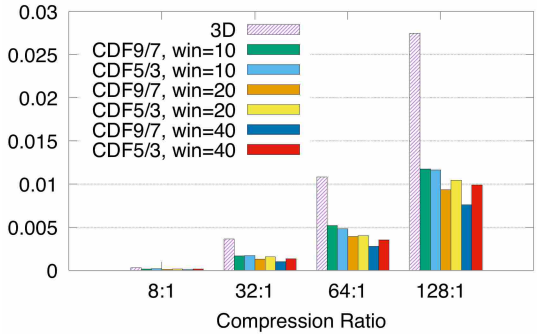
(a) Variable: enstrophy. Evaluation (Y-Axis): RMSE.



(b) Variable: enstrophy. Evaluation (Y-Axis): L^∞-norm.



(c) Variable: X-component of velocity. Evaluation (Y-Axis): RMSE.



(d) Variable: X-component of velocity. Evaluation (Y-Axis): L^∞-norm.

Figure 20. Evaluation on data from the Ghost simulation. Error values are normalized by the range of the data. Purple bars with line patterns represent spatial-only compression (3D), and bars with solid colors represent spatiotemporal compression (4D) with different parameters combinations for temporal compression. Two wavelet kernels (CDF9/7, CDF5/3) and three window sizes (win=10, 20, 40) are specifically examined here.

favorable combinations of the two. The experiments run were on data from the Ghost simulation, with the base temporal resolution.

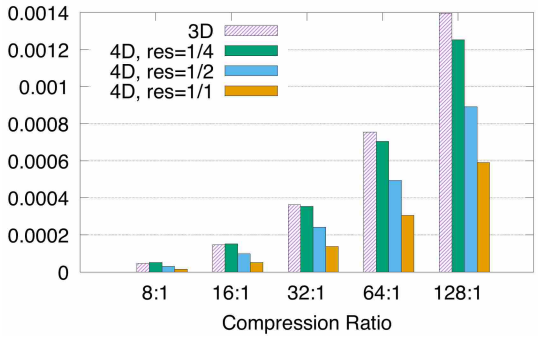
Figure 20 plot the evaluation results. Compression ratios are grouped together; spatial-only compression (3D) is leftmost within a group, and to its right are spatiotemporal compression with different parameters. All evaluations clearly show a decrease in error when comparing spatiotemporal to spatial-only compression.

The CDF 9/7 and CDF 5/3 wavelet kernels perform differently with different window sizes. CDF 9/7 yields lower errors than CDF 5/3 with window sizes 20 and 40, but CDF 5/3 is superior with window size 10. This is because, with a window size of 10, the CDF 9/7 kernel only permits one level of wavelet transform, while the CDF 5/3 allows for two levels.

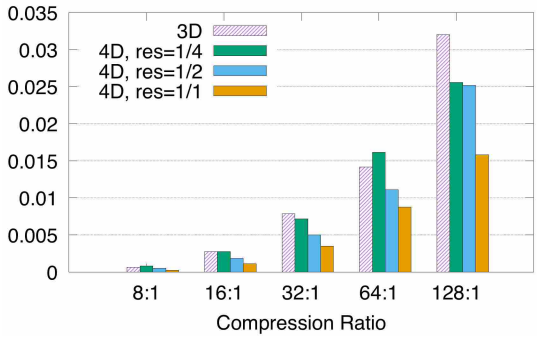
In terms of the window size, a larger window increases accuracy in almost every test. That said, the expected improvement from a larger window size varies. Errors drop more significantly when moving from window size 10 to 20 than moving from 20 to 40. Given the potential limitation from available memory, we consider the combination of windows size 20 and the CDF 9/7 wavelet kernel to be a "sweet-spot." A choice of CDF 5/3 and a window size of 10 for memory sparse situations would also make sense.

5.4.2.2 Temporal Resolution. This phase looked at the effects of temporal resolution. The experiments were run with the "sweet-spot" combination of wavelet kernel and window size from Section 5.4.2.1, again on the Ghost simulation.

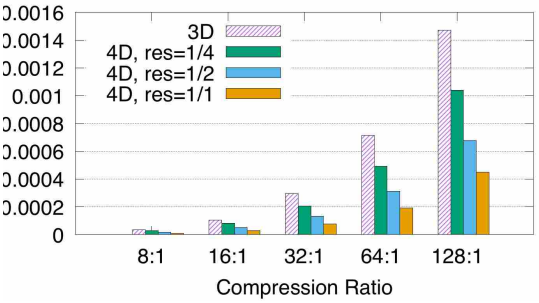
Figure 21 plots the results from this phase's experiments using NRMSE and normalized L^∞-norm metrics. Its 3D results are from all time slices at the



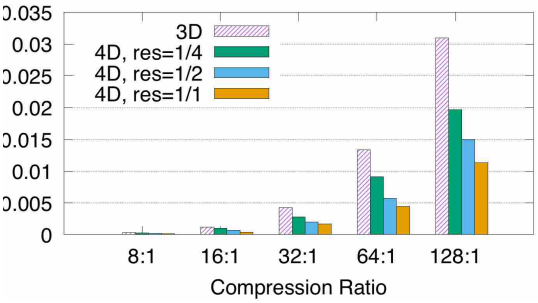
(a) Variable: enstrophy.
Evaluation (Y-Axis): RMSE.



(b) Variable: enstrophy.
Evaluation (Y-Axis): L^∞-norm.



(c) Variable: X-component of velocity.
Evaluation (Y-Axis): RMSE.



(d) Variable: X-component of velocity.
Evaluation (Y-Axis): L^∞-norm.

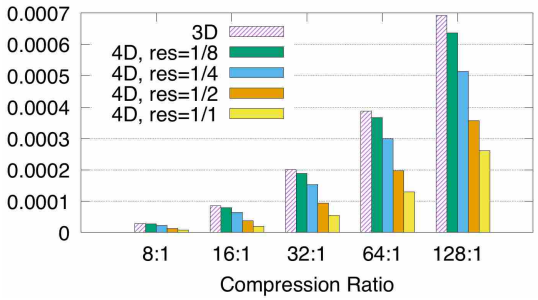
Figure 21. Evaluation on data from the Ghost simulation. Error values are normalized by the range of the data. Purple bars with line patterns represent spatial-only compression (3D), and bars with solid colors represent spatiotemporal compression (4D) with different parameters combinations for temporal compression. Three temporal resolutions (res=1/4, 1/2, 1/1) are specifically examined here.

base temporal resolution from the entire studied period of simulation. They serve as a baseline result to compare with. The 4D results are from multiple temporal resolutions (res=1, 1/2, 1/4). The 4D and 3D results cover the same period of simulation in each test.

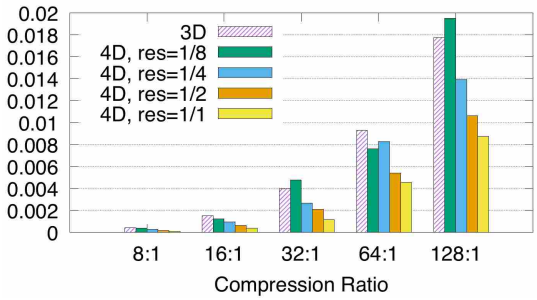
In terms of results, we see that the benefit of spatiotemporal compression improves as temporal resolution increases. At the finest resolution tested (res=1), spatiotemporal compression leads to substantial decrease in error compared to spatial compression. In most cases, both NRMSE and normalized L^∞-norm are cut by half when incorporating temporal compression. However, at the coarsest resolution tested (res=1/4), temporal compression brings modest benefit. In fact, the normalized L^∞-norm is even higher than the 3D baseline results in some cases (Subfigure 21b). We believe this is partially due to a lack of temporal coherence, and also partially due to the nature of L^∞-norm being more sensitive to single extreme values.

5.4.2.3 Results on Multiple Data Sets. This final phase of the initial study looked at the effectiveness of spatiotemporal compression on multiple data sets. The experiments were again run with the "sweet-spot" combination of wavelet kernel and window size from Section 5.4.2.1. Varying in this phase were both data set (see Section 5.4.1.3) and temporal resolution (see Section 5.4.1.2).

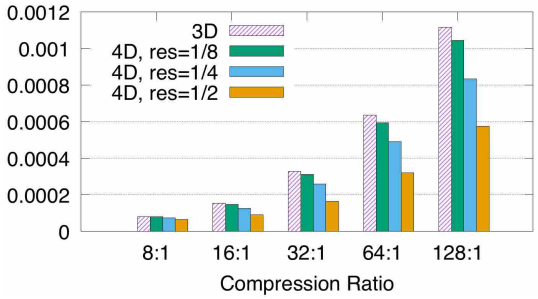
Figure 22 and Figure 23 plot the results from this phase's experiments using NRMSE and normalized L^∞-norm metrics. The former uses two variables from the CloverLeaf3D simulation, while the latter uses three variables from the Tornado simulations. These results confirm the effectiveness of spatiotemporal compression in most test cases, but also show that the amount of benefit relies on the temporal frequency of the data.



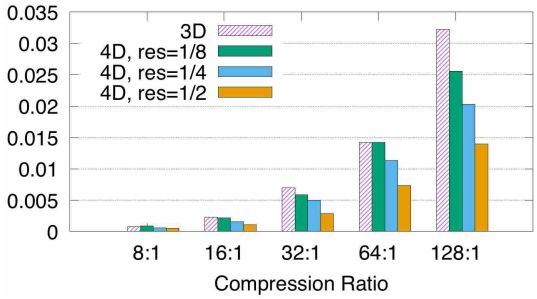
(a) Variable: X-component of velocity. Evaluation (Y-Axis): RMSE.



(b) Variable: X-component of velocity. Evaluation (Y-Axis): L^∞-norm.

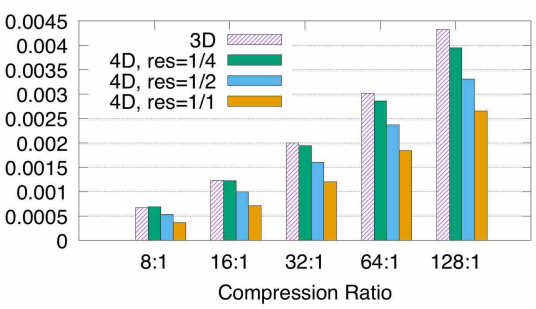


(c) Variable: energy. Evaluation (Y-Axis): RMSE.

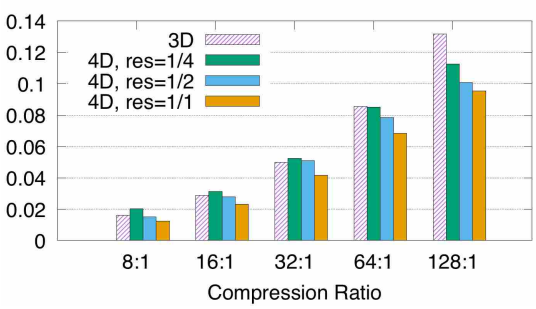


(d) Variable: energy. Evaluation (Y-Axis): L^∞-norm.

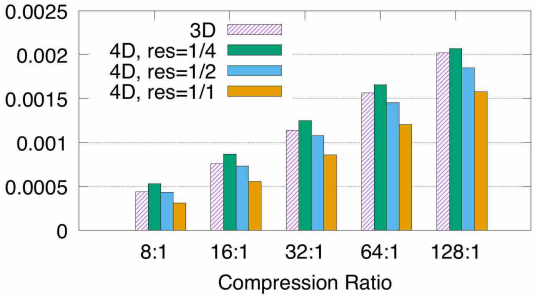
Figure 22. Evaluation results on variables from the CloverLeaf3D simulation (see Section 5.4.1.3). Error values are normalized by the range of the data. The purple bar with line patterns represents only spatial compression (3D), and the bars filled with solid colors represent spatiotemporal compression (4D) with different temporal resolutions.



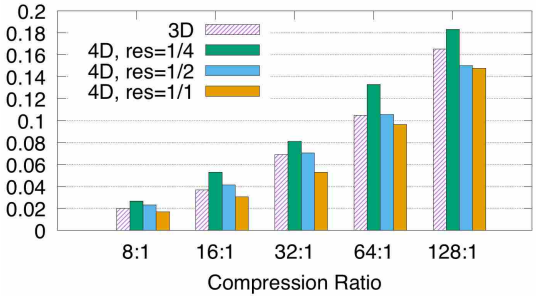
(a) Variable: X-component of velocity. Evaluation (Y-Axis): RMSE.



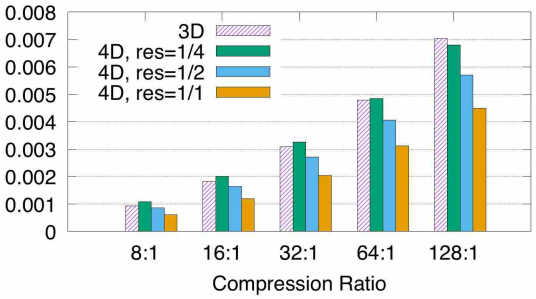
(b) Variable: X-component of velocity. Evaluation (Y-Axis): L^∞-norm.



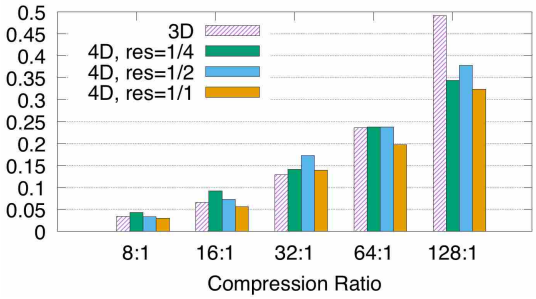
(c) Variable: enstrophy. Evaluation (Y-Axis): RMSE.



(d) Variable: enstrophy. Evaluation (Y-Axis): L^∞-norm.



(e) Variable: cloud ratio. Evaluation (Y-Axis): RMSE.



(f) Variable: cloud ratio. Evaluation (Y-Axis): L^∞-norm.

Figure 23. Evaluation on variables from the Tornado simulation (see Section 5.4.1.3). Error values are normalized by the range of the data. The purple bar with line patterns represents only spatial compression (3D), and the bars filled with solid colors represent spatiotemporal compression (4D) with different temporal resolutions.

Table 9. Performance impacts of spatiotemporal compression (4D) and spatial-only compression (3D) compared to no compression (Raw). The data set is the enstrophy field from the Ghost simulation. Error values are indicated by NRMSE that is reported in Subfigure 21a.

Tech.	Buffer W+R	Perm. Write	Total I/O	File Size	Comp. Time	Error
4D	6.78+6.5s	1.20s	14.48s	625MB	57.49s	5.18e-5
3D	0	1.20s	1.20s	625MB	55.34s	1.47e-4
Raw	0	18.90s	18.90s	10GB	0	0

5.4.3 Performance Impacts. Table 9 reports on performance impacts for spatiotemporal compression, as well as measurements for spatial-only and no compression. The column "Buffer W+R" indicates writing and reading time spent on the buffer space, "Perm. Write" indicates writing time spent on the permanent storage, and "Total I/O" indicates the sum of the buffer and permanent I/O costs. "Comp. Time" provides the computational cost. Our test system was a compute node with 2 Xeon CPUs at 3.2GHz (16 cores in total), 256GB main memory, and a 2TB SSD serving as a buffer space. The test data is 20 time slices of the enstrophy field from the Ghost simulation at the base temporal resolution (res=1/1). With each time slice at a 512^3 resolution, the total data size is approximately 10GB in raw format, and 625MB after a 16:1 compression (see the "File Size" column). Spatiotemporal compression here used the "sweet-spot" settings. Finally, note that for a specific grid resolution and number of time slices, the buffer I/O and computation cost numbers are independent of the data set and compression ratio — meaning that the results are applicable to data sets aside from Ghost and compression ratios aside from 16:1.

Compared to spatial-only compression, spatiotemporal compression introduces extra I/O time for buffer operations, and a modest amount of extra computation. In return, it encodes more information per byte. Compared to no

compression, spatiotemporal compression introduces reconstruction errors and additional computational burden. In return, it significantly reduces the size of data to save on permanent storage, and also saves on total I/O time (14.48s, compared to 18.90s).

5.4.4 Examining the Three Propositions. Section 5.1.2 introduced three propositions. These propositions are related, in that spatiotemporal compression provides more information per byte. That said, they attract domain scientists for different reasons. We examine each of these propositions separately here.

P1: *improve accuracy, while maintaining temporal resolution and storage costs.* **P1** is supported by our experiment results. For example, take the grouping for 128:1 compression in Subfigure 21c. In this grouping, the NRMSE for 3D compression is 1.47e-3, while 4D compression with sparse temporal sampling has NRMSE of 1.04e-3, and dense temporal sampling has NRMSE of 4.50e-4. In this case, then, 4D compression is anywhere from 40% more accurate to 200% more accurate for the same storage cost. The rest of our experiments provide similar results, except for the Tornado data set (Figure 23), which shows that the most sparse temporal sampling sometimes leads to slightly worse results (there is more discussion on the effect of temporal coherence in Section 5.4.5).

P2: *reduce storage costs, while maintaining temporal resolution and accuracy.* **P2** is also supported by our experiment results. Again consider an example from Subfigure 21c. The NRMSE with 64:1 compression with 3D wavelets is comparable to the error with 128:1 compression with 4D wavelets with "1/2" temporal resolution. In this case, a domain scientist could maintain accuracy and temporal resolution, but use half the storage. Similar examples

are visible throughout the table. However, the table is oriented around powers-of-two compression ratios, and **P2** does not always hold for $2X$ reductions in storage. Sticking with the example from Subfigure 21c, the NRMSE with 64:1 compression with 3D wavelets is more accurate than the 128:1 compression with "1/4" temporal resolution, which is coarser. In this case where time slices are sampled less frequently, a 2X reduction in storage was not obtained, but a smaller reduction (such as $1.5X$) likely would be possible.

P3: *increase temporal resolution, while maintaining storage costs and accuracy.* Proposition **P3** follows directly from **P2**: if it is possible to reduce storage costs and maintain accuracy, then it would also be possible to use the difference in storage to increase the temporal resolution. Revisiting the comparison between 3D+64:1 and 4D+128:1, a domain scientist could, instead of halving storage costs, opt to keep storage costs constant and double temporal resolution.

5.4.5 Discussions and Limitations. While we saw "factor of two" benefit in many cases (compared to spatial compression alone), other cases were below this amount. We believe limitations stem from an inadequate amount of temporal coherence between time slices to achieve good compression. On the one hand, the physical model and simulation implementation dictate the amount of coherence in the spatial domains. This means errors are lower in some applications, but higher in others. On the other hand, the output data frequency also plays an important role in temporal coherence. This means for the same application, spatiotemporal compression is more beneficial if time slices are sampled more frequently. Our tests across multiple data sets exhibit these limitations.

Among three tested simulations, we notice that Ghost and CloverLeaf3D have significantly less errors with the same spatial compression settings. For

example, looking at the X-velocity fields from three simulations, both Ghost and CloverLeaf3D have NRMSE less than 5e-5 at 8:1 ratio with spatial compression, but Tornado has NRMSE greater than 5e-4. That is one order of magnitude difference. Tornado has significantly larger normalized L^∞ norm values at all spatial compression cases as well. Since there is no temporal compression involved, we observe that it is the characteristic of the Tornado simulation that less coherence is present. In this case, wavelet-based compression techniques perform more poorly.

The accuracy increase from spatial to spatiotemporal compression also differs in the three simulations. Ghost and CloverLeaf3D see more than $2X$ accuracy increase (less than half of the error) at each compression level with the base temporal resolution (res=1/1). However, none of the three fields from Tornado see this amount of accuracy increase. We believe that this difference in accuracy gain is due to the difference of available temporal coherence: Ghost and CloverLeaf3D had enough temporal coherence to demonstrate a $2X$ accuracy gain, while Tornado was too sparse to do so. In fact, Subfigure 23c and 23e reveal that 4D compression even increases NRMSE errors at the coarsest resolution level (res=1/4). We suspect this is because the benefit from spatiotemporal compression is so modest in this instance that the floating point arithmetic errors from the additional calculations begin to dominate. More discussions on this topic can be found in (Keinert, 1995) and (Plonka, Schumacher, & Tasche, 2008). That said, spatiotemporal still shows benefit over spatial compression; the extra accuracy from spatiotemporal compression can still make a significant difference in real-world analyses, as we demonstrate in Section 5.5.1.

In terms of simulation scientists' control over the spatial and temporal coherence, we note that the spatial grid resolution is typically fixed by the properties of governing numerical equations. Thus the nature of the problem to be solved determines how strongly correlated samples are along a given spatial axis. The simulation scientist, however, may often easily control the output sampling rate — typically much coarser than the internal model time stepping — making the degree of temporal data coherence a parameter that can be readily adjusted.

Lastly, random access to individual time slices is easily lost during spatiotemporal compression. Specifically, during reconstruction of data from the compressed form, in the first step where inverse wavelet transforms are performed along the temporal domain, it needs to read in coefficients of other time slices that also belong to the same window. One can regain the ability of random access by employing smart coders on the resulting coefficients, for example, the one reported in (Rodler, 1999).

5.5 Applications to Real-World Analyses

In this section, we compare spatiotemporal (4D) wavelet compression with spatial-only (3D) wavelet compression on two real-world analyses. Both are representatives of regularly performed analyses by domain scientists, and they both operate on the Tornado simulation data set previously described in Section 5.4.1.3. The first analysis studied compression effects on pathlines across multiple time slices, while the second studied the effects on isosurfaces of a single time slice.

The mesh for the data set was a $490 \times 490 \times 280$ rectilinear grid, covering a geographic space of $14,670 \times 14,670 \times 8,370$ meters. The pathline analysis used data from 220 time slices, each stored as its own file. These 220 time slices were from the latter stages of the simulation when the tornado was most interesting to study.

Each time slice advanced two seconds of simulation time past the previous one, with the first time slice being 8502 seconds into the simulation — i.e., $t_0 = 8502s$, $t_1 = 8504s$, ..., $t_{219} = 8940s$. Note this temporal resolution corresponds to "res=1/2" from Section 5.4.1.2), which is not the finest available. We used this resolution since it is what our domain scientist collaborator uses in his own research.

5.5.1 Pathline Analysis.

5.5.1.1 Visualization. To better understand the tornado dynamics, we placed particles at the base of the tornado so that their trajectories could be observed. These particles are typically placed in a "rake" setting, i.e., densely seeding along a line segment. In this example, three rakes with 48 particles each were seeded, for a total of a 144 seed locations. The particles were advected using Runge-Kutta 4, with a step size of 0.01s. Velocity values between time slices were calculated using linear interpolation.

5.5.1.2 Compression. We worked with a total of nine versions of the data: the original version and eight wavelet-compressed versions. The wavelet-compressed versions included both spatiotemporal (4D) and spatial-only (3D) wavelet transforms over four compression levels (8:1, 32:1, 64:1, and 128:1). Both transforms used the CDF 9/7 wavelet kernel, and the spatiotemporal versions used a window size of 18. The three components of velocity were individually compressed for each wavelet version. We then generated a pathline for every combination of the 144 seed points and nine data sets, or $1,296$ pathlines overall. We set the baseline for each seed point as its pathline from the original version of the data set.

5.5.1.3 Evaluation. We defined our evaluation metric incorporating observations from visually comparing baseline pathlines with their versions from

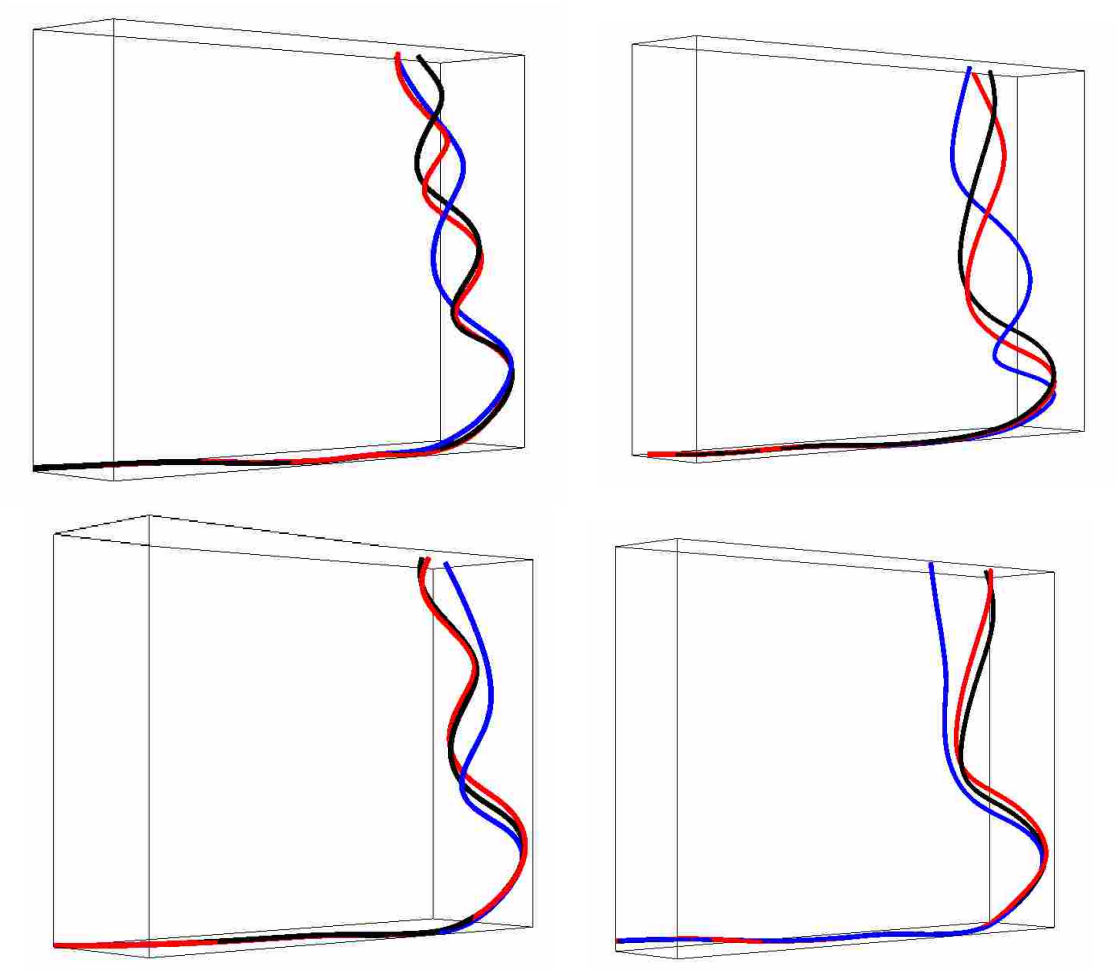


Figure 24. Each subfigure visualizes the pathlines for a single seed particle being advected using the original version of the data, as well as 3D and 4D compressed versions at 128:1 ratio. **Black** pathlines are from the original data set; **red** ones are from the 128:1+4D compression; and **blue** ones are from the 128:1+3D compression. The top two instances show that 4D (red) and 3D (blue) pathlines have similar ending positions, but the 4D ones closely resemble the baseline (black) for longer durations. The bottom two instances show a clear disadvantage of 3D pathlines in terms of both early deviation and far apart final positions.

Table 10. Our error metric for each wavelet-compressed data set, averaged over all 144 seed particles.

Data Set	D=10	D=50	D=150	D=300	D=500
8:1, 3D	8.5%	2.3%	1.3%	1.1%	1.0%
8:1, 4D	3.4%	1.3%	1.1%	0.8%	0.6%
32:1, 3D	35.9%	10.3%	4.5%	3.1%	2.4%
32:1, 4D	24.4%	6.4%	3.2%	2.1%	1.6%
64:1, 3D	48.4%	17.3%	**7.5%**	5.3%	3.9%
64:1, 4D	35.7%	9.8%	5.1%	3.3%	2.7%
128:1, 3D	60.7%	27.8%	10.8%	6.7%	5.2%
128:1, 4D	45.8%	16.3%	**7.5%**	5.1%	3.9%

wavelet-compressed data sets. Figure 24 visualizes example pathlines. Each image shows three pathlines generated by the advection of the same particle using the original and 3D or 4D compressed data at 128:1. Some particle trajectories would deviate midway through their courses, but then ultimately end up close to the correct positions, as the left two subfigures illustrate. So we designed an error metric that would value the case where a pathline stays close to its baseline throughout its entire trajectory, over one that deviates early but later returns. Specifically, let D be distance, let T be the total time of advection for a particle, and let T_0 be the first time that the pathline deviates distance D away from its baseline. Then we defined error as a percentage: $(1.0 - T_0/T) \times 100$. Taking an example: if a particle first deviates distance D from its baseline after six seconds and travels for a total of ten seconds, then we would score its error as 40%. We asked our domain scientist collaborator to select a good value for D and he picked a distance of 150 meters. We ran evaluations with that threshold, as well as bigger and smaller thresholds for comparative purposes. Five values of D were tested in our evaluation: D=10, 50, 150, 300, and 500.

5.5.1.4 Results. Table 10 contains the results from our evaluation. Each evaluation percentage is averaged from all 144 seed particles. This table shows a clear advantage of spatiotemporal compression, and supports both proposition **P1** and **P2**.

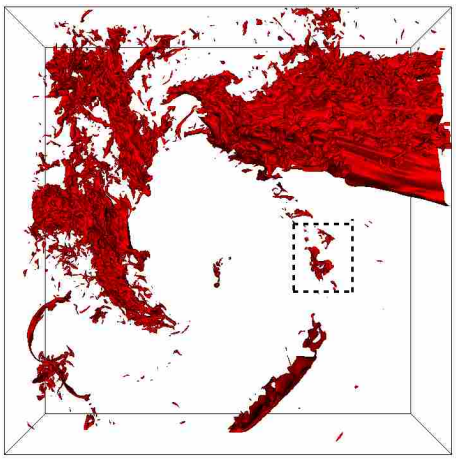
With respect to **P1** (improving accuracy while maintaining temporal resolution and storage costs), every combination of compression ratio and distance threshold shows the spatiotemporal compressed data to have superior accuracy (i.e., less error).

With respect to **P2** (reducing storage costs while maintaining temporal resolution and accuracy), comparisons between different compression ratios support this proposition. For example, for a distance threshold of 150, the 128:1+4D compression has the same error as the 64:1+3D compression (both are 7.5%, shown in bold font), meaning that the storage cost could be cut half. In fact, regardless of the distance threshold D, the error from 128:1+4D is always comparable to that from 64:1+3D, and the error from 64:1+4D is always comparable to that from 32:1+3D, supporting **P2** in a roughly $2X$ factor.

5.5.2 Isosurface Analysis.

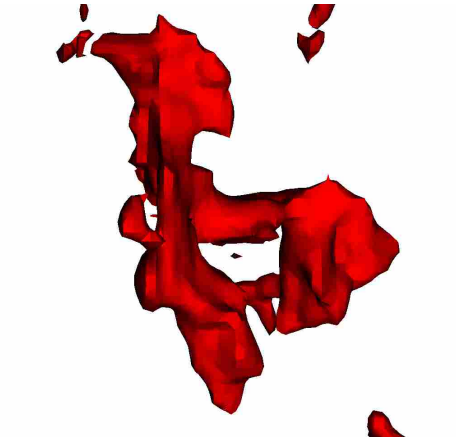
5.5.2.1 Visualization. Our domain scientist regularly studies isosurfaces in the Tornado data set. He provided us with three scalar variables that he often studies (pressure perturbation, cloud mixing ratio, and Z-component of velocity) as well as key isovalues appropriate for each of those variables.

5.5.2.2 Compression. We worked with a total of 33 versions of the data: the original version of three scalar fields as well as ten wavelet-compressed versions of each. The wavelet-compressed versions included both spatiotemporal (4D) and spatial-only (3D) wavelet transformations and five compression levels

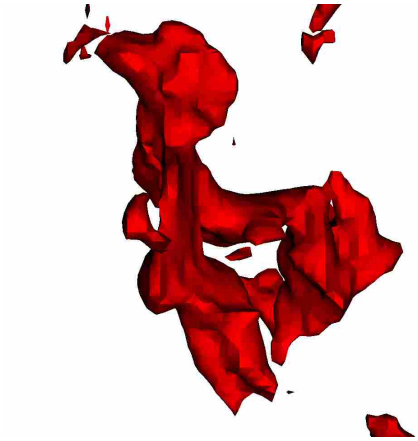


(a) Original data

(b) Original data



(c) 128:1+3D compression



(d) 128:1+4D compression

Figure 25. Renderings of isosurfaces of the z-component of velocity from a Tornado data set. Subfigure (a) is the entire data set, while (b), (c), and (d) are the same zoomed-in region. Dashed lines in (a) indicate this region.

(8:1, 16:1, 32:1, 64:1, and 128:1). We again used the CDF 9/7 kernel, and again the spatiotemporal versions used a window size of 18.

5.5.2.3 Evaluation. For each variable, we set the baseline isosurface to be the one from the original data set. Inspecting the isosurfaces visually, we found that the gross features from the baseline were well preserved within the wavelet-compressed versions, as Figure 25 shows. So our task shifted to capturing difference in fine details, and we opted to use the total surface area of the isosurfaces as our accuracy metric. That is, let SA_B be the surface area for the baseline isosurface and SA be the surface area for an isosurface from a wavelet-compressed data set. Then we defined our error metric again as a percentage: (1.0 - SA/SA_B) $\times$ 100. With this metric, 0% represents a perfect fit, with worse and worse fits moving away from 0 (in either the positive or negative directions). This metric creates the possibility for offsetting errors — compression may remove some features, but introduce others — but we did not observe this phenomenon in practice. Further, although we could have considered additional accuracy metrics (i.e., topological measures), we found that this simple metric corroborated our findings based on visual inspection, namely that 4D compression captures more surface details.

5.5.2.4 Results. Table 11 contains the results from our evaluation. Like the analysis from Section 5.5.1, this table supports both propositions **P1** and **P2**. With respect to **P1** (improving accuracy while maintaining temporal resolution and storage costs), the 4D compressed data had less error for every variable and compression level, except for the 8:1 compressed version of pressure perturbation (which had very small total error). As an example, for 32:1 compression on cloud mixing ratio, the isosurface with 3D compressed data was 4.72% too small (in terms of total surface area), but with 4D compressed data, the

Table 11. Our error metric for isosurfaces of each wavelet-compressed data set, comparing their surface area to the baseline surface area.

Variable	Compression	3D Error	4D Error
Cloud Mixing Ratio	8:1	-0.93%	0.35%
	16:1	-2.46%	0.59%
	32:1	-4.72%	0.47%
	64:1	-7.62%	-0.16%
	128:1	-11.24%	-1.34%
Z-Velocity	8:1	-0.85%	0.31%
	16:1	-2.23%	0.65%
	32:1	-4.69%	0.75%
	64:1	-8.85%	0.13%
	128:1	-15.320%	-1.66%
Pressure Perturbation	8:1	-2.4e-4%	-6.3e-3%
	16:1	-2.2e-2%	-1.8e-2%
	32:1	-4.9e-2%	-3.9e-2%
	64:1	-0.25%	-8.1e-2%
	128:1	-0.38%	6.7e-2%

isosurface was only 0.47% too big. Similar disparities held with other combinations of cloud mixing ratio and z-velocity. However, for pressure perturbation, both techniques seem to capture the isosurface, even at high compression ratios, and so neither seems superior to the other.

With respect to **P2** (reducing storage cost while maintaining accuracy and temporal resolution), this analysis and error metric strongly favor 4D wavelet compression over 3D wavelet compression. For example, our error metric finds that 128:1 4D compressed data is more accurate than 16:1 3D compressed data for Z-velocity.

5.6 Conclusions

Our study has considered the benefits, costs, and best practices for spatiotemporal wavelet compression compared to the traditional spatial-only approach. This direction is enabled by deeper memory hierarchies now

increasingly available on leading-edge supercomputers. We studied the benefits of spatiotemporal compression by looking at both differences in reconstructed fields over a variety of settings and several real-world analyses in consultation with domain scientists. In nearly all cases, we found that incorporating the time dimension led to more "information per byte," realized with a variety of error metrics: NRMSE, normalized L^∞, and custom metrics for real-world applications. This property in turn enabled three propositions that can benefit domain scientists with their visualization needs — improving on accuracy, reducing storage, and increasing temporal frequency. While the magnitude of the benefit varies by use case, many of the results demonstrated factor-of-two improvements for their respective metrics.

Finally, unlike spatial coherence, temporal coherence is a function of how frequently the data is output. Often output frequency is constrained by storage space, and if too coarse, benefits from temporal compression may be small. That said, there are many analyses where higher output cadence is required, and spatiotemporal compression shows great promise when faced with constraints of limited I/O bandwidth and storage capacity.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The growing discrepancy between I/O and computation on HPC systems is one of the major challenges simulation scientists face today. As this trend continues, simulation scientists will have to accept data reduction for part, if not all, of the application scenarios. Though there are many data reduction techniques available, few are studied thoroughly, especially their efficacy for real visualization and analysis tasks. This dissertation has investigated one of the lossy compression techniques, namely wavelet compression, in a high performance computing environment. Specifically, there are three research questions this dissertation has investigated with a "yes" answer: 1) whether the data integrity is still acceptable after lossy compression; 2) whether wavelet compression is viable for in situ compression; and 3) whether wavelet compression can adapt to emerging HPC architectures. We also identified the most suitable wavelet parameters to carry out compression tasks on scientific data sets along with our study. Findings from this dissertation support that lossy wavelet compression can be used as an in situ compressor on supercomputers to mitigate I/O bottlenecks.

6.2 Future Work

There are three interesting future directions from this dissertation.

The first direction is the integration of an in situ wavelet compressor with real large-scale simulations in a production setting. This integration should be performed together with domain scientists, in order to make sure their concerns are properly identified and addressed. Looking at even larger simulations, for example the ones on Exascale HPC systems, there are always challenges coming

with the increased scale of problems. Some of these challenges are hard to predict before actually running experiments. We would effectively explore these challenges through large-scale experiment runs, and also address any integration issues for Exascale applications.

The second direction is to keep improving the wavelet compressor itself. There are a few possible improvements, including the use of advanced coders and providing a mode for bounding error.

Advanced coders could eliminate the addressing cost of the coefficient prioritization strategy, providing better compression results. Two such coders that are promising for our use are SPECK and SPIHT, as discussed in Section 2.2.4.2. With a potential introduction of these coders, additional work is then required to understand their characteristics such as performance and adaptation to modern architectures.

We also believe an error bound mode would provide important capability for domain scientists, by guaranteeing an error tolerance, although not a storage budget. This mode contrasts with the traditional approach, which guarantees a storage budget but not an error tolerance. An error bound mode is desirable for many scientists because it controls how far away each value can be from its original value, allowing scientists to feel confident in the resulting analyses. As a transform-based technique, wavelet compression does not natively support error bounding. Special tweaks to the algorithms are required to make it happen, as Lindstrom et al. demonstrated in the case of ZFP (Lindstrom, 2014; Lindstrom, Chen, & Lee, 2016), another transform-based compression technique. Again, additional work is required to assess its viability.

The third direction is to compare with other compressors and better understand their advantages and disadvantages. We believe this is necessary to deploy in situ compression across multiple applications, because a wide range of applications may require compressors with diverse characteristics. For example, wavelet compressor works well on "smooth" data while another might be more suitable for data with another characteristic. It is even possible to automate the process of selecting the most suitable compressor for any incoming data. This goal is ambitious since it requires good understanding on not only one, but many, compressor candidates. It is also rewarding since it has the potential to achieve globally optimal compression results.

REFERENCES CITED

Acharya, T., & Tsai, P.-S. (2004). *JPEG2000 standard for image compression: Concepts, algorithms and VLSI architectures*. Wiley-Interscience.

Agranovsky, A., Camp, D., Garth, C., Bethel, E. W., Joy, K. I., & Childs, H. (2014, November). Improved Post Hoc Flow Analysis Via Lagrangian Representations. In *Proceedings of the ieee symposium on large data visualization and analysis (ldav)* (pp. 67–75). Paris, France.

Ahrens, J., Jourdain, S., O'Leary, P., Patchett, J., Rogers, D. H., & Petersen, M. (2014). An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 424–434).

Antonini, M., Barlaud, M., Mathieu, P., & Daubechies, I. (1992). Image coding using wavelet transform. *IEEE Transactions on Image Processing*, *1*(2), 205–220.

Ao, J., Mitra, S., & Nutter, B. (2014). Fast and efficient lossless image compression based on CUDA parallel wavelet tree encoding. In *Ieee southwest symposium on image analysis and interpretation (ssiai)* (pp. 21–24).

Baker, A. H., Xu, H., Dennis, J. M., Levy, M. N., Nychka, D., Mickelson, S. A., ... Wegener, A. (2014). A methodology for evaluating the impact of data compression on climate simulation data. In *Proceedings of the 23rd international symposium on high-performance parallel and distributed computing* (pp. 203–214).

Balsa Rodríguez, M., Gobbetti, E., Iglesias Guitián, J., Makhinya, M., Marton, F., Pajarola, R., & Suter, S. K. (2014). State-of-the-art in compressed GPU-based direct volume rendering. In *Computer graphics forum* (Vol. 33, pp. 77–100).

Bauer, A. C., Abbasi, H., Ahrens, J., Childs, H., Geveci, B., Klasky, S., ... Bethel, E. W. (2016). In situ methods, infrastructures, and applications on high performance computing platforms. In *Proceedings of the eurographics: State of the art reports* (pp. 577–597). Goslar, Germany: Eurographics Association.

Bell, N., & Hoberock, J. (2011). Thrust: A productivity-oriented library for CUDA. *GPU computing gems Jade edition*, *2*, 359–371.

Bertram, M., Duchaineau, M. A., Hamann, B., & Joy, K. I. (2000). Bicubic subdivision-surface wavelets for large-scale isosurface representation and visualization. In *Proceedings of the conference on visualization'00* (pp. 389–396).

Blelloch, G. E. (1990). *Vector models for data-parallel computing*. MIT press Cambridge.

Bosi, M., Brandenburg, K., Quackenbush, S., Fielder, L., Akagiri, K., Fuchs, H., & Dietz, M. (1997). ISO/IEC MPEG-2 advanced audio coding. *Journal of the Audio engineering society*, *45*(10), 789–814.

Burrus, C. S., Gopinath, R. A., Guo, H., Odegard, J. E., & Selesnick, I. W. (1998). *Introduction to wavelets and wavelet transforms: a primer*. Prentice hall New Jersey.

Burtscher, M., & Ratanaworabhan, P. (2009). FPC: A high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers*, *58*(1), 18–31.

Carr, H. A., Weber, G. H., Sewell, C. M., & Ahrens, J. P. (2016). Parallel peak pruning for scalable SMP contour tree computation. In *Ieee the 6th symposium on large data analysis and visualization (ldav)* (p. 75-84).

Chadha, N., Cuhadar, A., & Card, H. (2002). Scalable parallel wavelet transforms for image processing. In *Canadian conference on electrical and computer engineering, 2002. (ccece)* (Vol. 2, pp. 851–856).

Chaver, D., Prieto, M., Pinuel, L., & Tirado, F. (2002, April). Parallel wavelet transform for large scale image processing. In *Proceedings 16th international parallel and distributed processing symposium* (p. 6 pp-).

Childs, H., Brugger, E., Whitlock, B., Meredith, J., Ahern, S., Pugmire, D., ... Navrátil, P. (2012). VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight* (p. 357-372).

CISL. (2017). Cheyenne: SGI ICE XA System.. doi: 10.5065/D6RX99HX

Clyne, J., Mininni, P., Norton, A., & Rast, M. (2007). Interactive desktop analysis of high resolution simulations: application to turbulent plume dynamics and current sheet formation. *New Journal of Physics*, *9*(8), 301.

Clyne, J., & Rast, M. (2005). A prototype discovery environment for analyzing and visualizing terascale turbulent fluid flow simulations. In *Electronic imaging 2005* (pp. 284–294).

Cohen, A., Daubechies, I., & Feauveau, J.-C. (1992). Biorthogonal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, *45*(5), 485–560.

Di, S., & Cappello, F. (2016). Fast error-bounded lossy hpc data compression with SZ. In *2016 ieee international parallel and distributed processing symposium (ipdps)* (p. 730-739).

Enfedaque, P., Auli-Llinas, F., & Moure, J. C. (2015). Implementation of the DWT in a GPU through a register-based strategy. *IEEE Transactions on Parallel and Distributed Systems*, *26*(12), 3394–3406.

Fabian, N., Moreland, K., Thompson, D., Bauer, A. C., Marion, P., Gevecik, B., ... Jansen, K. E. (2011). The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *Ieee symposium on large data analysis and visualization (ldav)* (pp. 89–96).

Franco, J., Bernabé, G., Fernández, J., & Ujaldón, M. (2010). Parallel 3D fast wavelet transform on manycore GPUs and multicore CPUs. *Procedia Computer Science*, *1*(1), 1101–1110.

Gaither, K. P., Childs, H., Schulz, K. W., Harrison, C., Barth, W., Donzis, D., & Yeung, P.-K. (2012). Visual Analytics for Finding Critical Structures in Massive Time-Varying Turbulent-Flow Simulations. *IEEE Computer Graphics and Applications*, *32*(4), 34–45.

Garcia, A., & Shen, H.-W. (2005). GPU-based 3D wavelet reconstruction with tileboarding. *The Visual Computer*, *21*(8), 755–763.

Gioia, P., Aubault, O., & Bouville, C. (2004). Real-time reconstruction of wavelet-encoded meshes for view-dependent transmission and visualization. *IEEE Transactions on Circuits and Systems for Video Technology*, *14*(7), 1009–1020.

Gruchalla, K., Rast, M., Bradley, E., Clyne, J., & Mininni, P. (2009). Visualization-driven structural and statistical analysis of turbulent flows. In *Advances in intelligent data analysis viii* (pp. 321–332). Springer.

Guthe, S., & Straßer, W. (2001). Real-time decompression and visualization of animated volume data. In *Visualization, 2001. vis'01. proceedings* (pp. 349–572).

Haar, A. (1910). Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, *69*(3), 331–371.

Heiland, R., & Baker, M. P. (1998). A survey of co-processing systems. *CEWES MSRC PET Technical Report*, 98–52.

Ibarria, L., Lindstrom, P., Rossignac, J., & Szymczak, A. (2003). Out-of-core compression and decompression of large n-dimensional scalar fields. In *Computer graphics forum* (Vol. 22, pp. 343–348).

Ihm, I., & Park, S. (1999). Wavelet-based 3D compression scheme for interactive visualization of very large volume data. In *Computer graphics forum* (Vol. 18, pp. 3–15).

Islam, A., & Pearlman, W. A. (1998). Embedded and efficient low-complexity hierarchical image coder. In *Electronic imaging'99* (pp. 294–305).

Karlin, I., Keasler, J., & Neely, R. (2013, August). *Lulesh 2.0 updates and changes* (Tech. Rep. No. LLNL-TR-641973).

Keinert, F. (1995). Numerical stability of biorthogonal wavelet transforms. *Advances in Computational Mathematics*, *4*(1), 1–26.

Kim, B.-J., & Pearlman, W. A. (1997). An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPIHT). In *Proceedings of the data compression conference, dcc'97.* (pp. 251–260).

Krajcevski, P., Pratapa, S., & Manocha, D. (2016). GST: GPU-decodable supercompressed textures. *ACM Transactions on Graphics (TOG)*, *35*(6), 230.

Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R., & Samatova, N. F. (2011). Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In *Euro-par 2011 parallel processing* (pp. 366–379). Springer.

Lalgudi, H. G., Bilgin, A., Marcellin, M. W., & Nadar, M. S. (2005). Compression of fMRI and ultrasound images using 4D SPIHT. In *Ieee international conference on image processing* (Vol. 2, pp. II–746).

Lalgudi, H. G., Bilgin, A., Marcellin, M. W., Tabesh, A., Nadar, M. S., & Trouard, T. P. (2005). Four-dimensional compression of fMRI using JPEG2000. In *Proceedings of spie* (Vol. 5747, p. 1029).

Laney, D., Langer, S., Weber, C., Lindstrom, P., & Wegener, A. (2013). Assessing the effects of data compression in simulations using physically motivated metrics. In *Proceedings of sc13: International conference for high performance computing, networking, storage and analysis* (p. 76).

Larsen, M., Aherns, J., Ayachit, U., Brugger, E., Childs, H., Geveci, B., & Harrison, C. (2017). The alpine in situ infrastructure: Ascending from the ashes of strawman. In *Proceedings of the in situ infrastructures for enabling extreme-scale analysis and visualization workshop.* New York, NY, USA: ACM.

Larsen, M., Labasan, S., Navrátil, P., Meredith, J., & Childs, H. (2015, May). Volume Rendering Via Data-Parallel Primitives. In *Proceedings of eurographics symposium on parallel graphics and visualization (egpgv)* (p. 53-62). Cagliari, Italy.

Larsen, M., Meredith, J., Navrátil, P., & Childs, H. (2015, April). Ray-Tracing Within a Data Parallel Framework. In *Proceedings of the ieee pacific visualization symposium* (pp. 279–286). Hangzhou, China.

Larsen, M., et al. (2015). Strawman: A batch in situ visualization and analysis infrastructure for multi-physics simulation codes. In *Proceedings of the workshop on in situ infrastructures for enabling extreme-scale analysis and visualization (isav)* (pp. 30–35). New York, NY, USA: ACM.

Lehmann, H., & Jung, B. (2014). In-situ multi-resolution and temporal data compression for visual exploration of large-scale scientific simulations. In *2014 ieee 4th symposium on large data analysis and visualization (ldav)* (pp. 51–58).

Lessley, B., Binyahib, R., Maynard, R., & Childs, H. (2016). External Facelist Calculation with Data-Parallel Primitives. In *Proceedings of eurographics symposium on parallel graphics and visualization (egpgv)* (p. 10-20). Groningen, The Netherlands.

Li, S., Gruchalla, K., Potter, K., Clyne, J., & Childs, H. (2015). Evaluating the Efficacy of Wavelet Configurations on Turbulent-Flow Data. In *Proceedings of ieee symposium on large data analysis and visualization (ldav)* (pp. 81–89). Chicago, IL.

Li, S., Larsen, M., Clyne, J., & Childs, H. (2017). Performance impacts of in situ wavelet compression on scientific simulations. In *Proceedings of the in situ infrastructures on enabling extreme-scale analysis and visualization* (pp. 37–41). New York, NY, USA: ACM.

Li, S., Marsaglia, N., Chen, V., Sewell, C., Clyne, J., & Childs, H. (2017). Achieving Portable Performance For Wavelet Compression Using Data Parallel Primitives. In *Eurographics symposium on parallel graphics and visualization.* The Eurographics Association.

Li, S., Sane, S., Orf, L., Mininni, P., Clyne, J., & Childs, H. (2017). Spatiotemporal wavelet compression for visualization of scientific simulation data. In *2017 ieee international conference on cluster computing (cluster)* (p. 216-227).

Liang, J.-Y., Chen, C.-S., Huang, C.-H., & Liu, L. (2008). Lossless compression of medical images using hilbert space-filling curves. *Computerized Medical Imaging and Graphics*, *32*(3), 174–182.

Lindstrom, P. (2014). Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, *20*(12), 2674–2683.

Lindstrom, P., Chen, P., & Lee, E.-J. (2016). Reducing disk storage of full-3d seismic waveform tomography (f3dt) through lossy online compression. *Computers & Geosciences*, *93*, 45–54.

Lindstrom, P., & Isenburg, M. (2006). Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, *12*(5), 1245–1250.

Liu, N., Cope, J., Carns, P., Carothers, C., Ross, R., Grider, G., ... Maltzahn, C. (2012). On the role of burst buffers in leadership-class storage systems. In *Ieee 28th symposium on mass storage systems and technologies (msst)* (pp. 1–11).

Lo, L.-t., Sewell, C., & Ahrens, J. (2012). PISTON: A Portable Cross-Platform Framework for Data-Parallel Visualization Operators. In H. Childs, T. Kuhlen, & F. Marton (Eds.), *Eurographics symposium on parallel graphics and visualization.* The Eurographics Association.

Mallinson, A., Beckingsale, D. A., Gaudin, W., Herdman, J., Levesque, J., & Jarvis, S. A. (2013). Cloverleaf: Preparing hydrodynamics codes for exascale. *The Cray User Group*, 6–9.

Maynard, R., Moreland, K., Atyachit, U., Geveci, B., & Ma, K.-L. (2013). Optimizing threshold for extreme scale analysis. In (Vol. 8654, p. 8654 - 8654 - 8).

Mininni, P., Alexakis, A., & Pouquet, A. (2006). Large-scale flow effects, energy transfer, and self-similarity on turbulence. *Physical Review E*, *74*(1), 016303.

Mininni, P., Alexakis, A., & Pouquet, A. (2008). Nonlocal interactions in hydrodynamic turbulence at high reynolds numbers: The slow emergence of scaling laws. *Physical review E*, *77*(3), 036306.

Moreland, K., Sewell, C., Usher, W., t. Lo, L., Meredith, J., Pugmire, D., ... Geveci, B. (2016, May). Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications*, *36*(3), 48-58.

Morton, G. M. (1966). *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company.

Mulder, J. D., van Wijk, J. J., & van Liere, R. (1999). A survey of computational steering environments. *Future Generation Computer Systems*, *15*(1), 119 - 129.

Nickolls, J., Buck, I., Garland, M., & Skadron, K. (2008, March). Scalable parallel programming with CUDA. *Queue*, *6*(2), 40–53.

Nielsen, O. M., & Hegland, M. (2000). Parallel performance of fast wavelet transforms. *International Journal of High Speed Computing*, *11*(01), 55–74.

Norton, A., & Clyne, J. (2012). The VAPOR visualization application. In E. Bethel, H. Childs, & C. Hanson (Eds.), *High performance visualization* (p. 415-428).

Nouanesengsy, B., Woodring, J., Patchett, J., Myers, K., & Ahrens, J. (2014). Adr visualization: A generalized framework for ranking large-scale scientific data using analysis-driven refinement. In *2014 ieee 4th symposium on large data analysis and visualization (ldav)* (pp. 43–50).

Nvidia, C. (2015). *Achieved occupancy.* Retrieved 2017-03-04, from `https://goo.gl/xHfGlO`

Nyquist, H. (1928). Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, *47*(2), 617–644.

Olanda, R., Pérez, M., Orduña, J. M., & Rueda, S. (2014). Terrain data compression using wavelet-tiled pyramids for online 3D terrain visualization. *International Journal of Geographical Information Science*, *28*(2), 407–425.

Orf, L., Wilhelmson, R., Lee, B., Finley, C., & Houston, A. (2017). Evolution of a long-track violent tornado within a simulated supercell. *Bulletin of the American Meteorological Society*, *98*(1), 45-68.

Orf, L., Wilhelmson, R., & Wicker, L. (2016). Visualization of a simulated long-track EF5 tornado embedded within a supercell thunderstorm. *Parallel Computing*, *55*, 28–34.

Pearlman, W. A., Islam, A., Nagaraj, N., & Said, A. (2004). Efficient, low-complexity image coding with a set-partitioning embedded block coder. *IEEE Transactions on Circuits and Systems for Video Technology*, *14*(11), 1219–1235.

Pheatt, C. (2008). Intel® threading building blocks. *Journal of Computing Sciences in Colleges*, *23*(4), 298–298.

Plonka, G., Schumacher, H., & Tasche, M. (2008). Numerical stability of biorthogonal wavelet transforms. *Advances in Computational Mathematics*, *29*(1), 1–25.

Rodler, F. F. (1999). Wavelet based 3D compression with fast random access for very large volume data. In *Seventh pacific conference on computer graphics and applications* (pp. 108–117).

Sakai, R., Sasaki, D., Obayashi, S., & Nakahashi, K. (2013). Wavelet-based data compression for flow simulation on block-structured cartesian mesh. *International Journal for Numerical Methods in Fluids*, *73*(5), 462–476.

Sanchez, V., Nasiopoulos, P., & Abugharbieh, R. (2008). Efficient 4D motion compensated lossless compression of dynamic volumetric medical image data. In *Ieee international conference on acoustics, speech and signal processing, icassp 2008* (pp. 549–552).

Schroots, H. A., & Ma, K.-L. (2015). Volume Rendering with Data Parallel Visualization Frameworks for Emerging High Performance Computing Architectures. In *Siggraph asia 2015 visualization in high performance computing* (pp. 3:1–3:4). ACM.

Scivoletto, A., & Romano, N. (2016). Performances of a parallel cuda program for a biorthogonal wavelet filter. In *Proceedings of the international symposium for young scientists in technology, engineering and mathematics (system).*

Skodras, A., Christopoulos, C., & Ebrahimi, T. (2001). The JPEG2000 still image compression standard. *IEEE Signal processing magazine*, *18*(5), 36–58.

Stollnitz, E. J., DeRose, T. D., & Salesin, D. H. (1996). *Wavelets for computer graphics: theory and applications.* Morgan Kaufmann.

Strang, G., & Nguyen, T. (1996). *Wavelets and filter banks.* SIAM.

Sweldens, W. (1996). The lifting scheme: A custom-design construction of biorthogonal wavelets. *Applied and computational harmonic analysis*, *3*(2), 186–200.

Tang, X., Pearlman, W. A., & Modestino, J. W. (2003). Hyperspectral image compression using three-dimensional wavelet coding. In *Electronic imaging 2003* (pp. 1037–1047).

Tao, D., Di, S., Chen, Z., & Cappello, F. (2017). Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. *Proceedings of the 31th IEEE International Parallel and Distributed Processing Symposium*.

Tao, H., & Moorhead, R. J. (1994). Progressive transmission of scientific data using biorthogonal wavelet transform. In *Proceedings of the conference on visualization'94* (pp. 93–99).

Taubman, D. (2000). High performance scalable image compression with EBCOT. *IEEE transactions on Image Processing*, *9*(7), 1158–1170.

Trott, A., Moorhead, R., & McGinley, J. (1996). Wavelets applied to lossless compression and progressive transmission of floating point data in 3D curvilinear grids. In *Proceedings of visualization'96.* (pp. 385–388).

Uhl, A. (1995). A parallel wavelet image block-coding algorithm. In *Proceedings of intern. conference on high performance computing.*

Unser, M., & Blu, T. (2003). Mathematical properties of the JPEG2000 wavelet filters. *IEEE Transactions on Image Processing*, *12*(9), 1080–1090.

Usevitch, B. E. (2001). A tutorial on modern lossy wavelet image compression: foundations of jpeg 2000. *Signal Processing Magazine, IEEE*, *18*(5), 22–35.

Villasenor, J. D., Belzer, B., & Liao, J. (1995). Wavelet filter evaluation for image compression. *IEEE Transactions on Image Processing*, *4*(8), 1053–1060.

Villasenor, J. D., Ergas, R., & Donoho, P. (1996). Seismic data compression using high-dimensional wavelet transforms. In *Proceedings of the data compression conference, dcc'96.* (pp. 396–405).

Wang, C., Gao, J., Li, L., & Shen, H.-W. (2005). A multiresolution volume rendering framework for large-scale time-varying data visualization. In *Fourth international workshop on volume graphics* (pp. 11–223).

Wang, T., Oral, S., Wang, Y., Settlemyer, B., Atchley, S., & Yu, W. (2014). Burstmem: A high-performance burst buffer system for scientific applications. In *Ieee international conference on big data (big data)* (pp. 71–79).

Whitlock, B., Favre, J. M., & Meredith, J. S. (2011). Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th eurographics conference on parallel graphics and visualization* (pp. 101–109).

Williams, L. (1983). Pyramidal parametrics. In *Acm siggraph computer graphics* (Vol. 17, pp. 1–11).

Woodring, J., Mniszewski, S., Brislawn, C., DeMarle, D., & Ahrens, J. (2011). Revisiting wavelet compression for large-scale climate data using JPEG 2000 and ensuring data precision. In *Ieee symposium on large data analysis and visualization (ldav)* (p. 31-38).

Xie, B., Chase, J., Dillow, D., Drokin, O., Klasky, S., Oral, S., & Podhorszki, N. (2012). Characterizing output bottlenecks in a supercomputer. In *Proceedings of the international conference on high performance computing, networking, storage and analysis* (p. 8).

Zajac, E. E. (1964, March). Computer-made perspective movies as a scientific and communication tool. *Commun. ACM*, *7*(3), 169–170.

Zeng, L., Jansen, C., Unser, M. A., & Hunziker, P. (2001). Extension of wavelet compression algorithms to 3D and 4D image data: Exploitation of data coherence in higher dimensions allows very high compression ratios. In *International symposium on optical science and technology* (pp. 427–433).

Zeng, L., Jansen, C. P., Marsch, S., Unser, M., & Hunziker, P. R. (2002). Four-dimensional wavelet compression of arbitrarily sized echocardiographic data. *IEEE Transactions on Medical Imaging*, *21*(9), 1179–1187.