CUDA-ACCELERATED HD-ODETLAP: A HIGH DIMENSIONAL GEOSPATIAL DATA COMPRESSION FRAMEWORK

By

You Li

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Approved by the
Examining Committee:

_____
W. Randolph Franklin, Thesis Adviser

_____
Mukkai S. Krishnamoorthy, Member

_____
Peter Fox, Member

_____
Michael J. Wozny, Member

Rensselaer Polytechnic Institute
Troy, New York

July 2011
(For Graduation August 2011)

CONTENTS

LIST OF TABLES iii

LIST OF FIGURES vi

ACKNOWLEDGMENT ix

ABSTRACT x

1. INTRODUCTION 1
   - 1.1 Motivation 1
   - 1.2 HD-ODETLAP 4
   - 1.3 Thesis Outline 5

2. Background 7
   - 2.1 General Compression Methods 7
   - 2.2 ODETLAP 9
     - 2.2.1 ODETLAP Framework 9
     - 2.2.2 ODETLAP Slope Compression 11
     - 2.2.3 Parallel ODETLAP 11
   - 2.3 Linear Solution in CUDA 12
     - 2.3.1 GPGPU background 12
     - 2.3.2 CUDA programming architecture 12
     - 2.3.3 Motivation 14

3. HD-ODETLAP 18
   - 3.1 3D-ODETLAP 18
     - 3.1.1 Definition 18
     - 3.1.2 Algorithm Outline 19
     - 3.1.3 Forbidden Zone 21
     - 3.1.4 Implementation Speed-up 22
       - 3.1.4.1 Normal Equation 22
       - 3.1.4.2 Dividing into Boxes 22
     - 3.1.5 Further Compression-Run Length Encoding 25
   - 3.2 5D-ODETLAP 26
     - 3.2.1 Definition 26

3.2.2 Framework Outline . . . 28
3.2.3 Four Dimension Forbidden Zone . . . 30
3.2.4 Four Dimension Dividing into Boxes . . . 31
3.2.5 Initial Point Selection . . . 31
3.2.6 Further Encoding . . . 32
3.2.6.1 Algorithm Outline . . . 32
3.2.6.2 Binary Position Compression . . . 34
3.2.6.3 Floating Value Compression . . . 35
4. CUDA SOLVER FOR HD-ODETLAP . . . 45
4.1 Solver Introduction . . . 45
4.2 Solver Selection and Matlab Integration . . . 46
4.3 Residual Size Test . . . 51
5. RESULTS AND ANALYSIS . . . 54
5.1 3D-ODETLAP . . . 54
5.1.1 Results and Analysis . . . 54
5.1.2 Comparison with 3D-SPIHT . . . 57
5.2 Preliminary 5D-ODETLAP . . . 60
5.2.1 Results and Analysis . . . 60
5.2.2 Comparison with 3D-SPIHT . . . 63
5.3 Optimized 5D-ODETLAP Results and Comparison . . . 65
6. CONCLUSION . . . 73
7. FUTURE WORK . . . 75
LITERATURE CITED . . . 76
APPENDICES
A. Sample CUSP code . . . 83
B. Sample code in mixed CUSP and MEX style . . . 85

LIST OF TABLES

3.1 File size comparison after compression between the Run length encoding method and simply storing quadruplets which contains three coordinates and values. "Compr.xyz(RLE)" is the size of compressed coordinates using RLE and bizp2. "Compr.v" is the size of compressed scalar value using RLE and bizp2. "Bzip2" is the size of compressed 3D dataset using only bizp2. Our RLE method further reduces file size ranging from 12% to 34% as indicated in the fourth row 25

3.2 Results of initial approximation from using different initial point selection methods on two datasets. Data1 has 99.82% filled points and Data2 has only 58.01% filled points. Random1 and Random2 are two runs on the same dataset using Matlab's uniformly random method. . . 31

3.3 Influence of delta encoding on the size of the final compressed floating point value vector. Column "Advantage" indicates the advantage of delta encoding over non-delta encoding on the final result. Test data is the same as in Figure 3.11. 36

4.1 Efficiency improvement of overhead-free CUSP CG solver over Matlab CF direct solver using Cholesky Factorization. Elapsed time measurement is in seconds. Size of the data represents the size of the linear system from the 5D-ODETLAP method. The linear systems solved are constructed from the datasets in the table in 4D reconstruction of the 5D-ODETLAP method. 48

4.2 Elapsed time (in seconds) comparison between Matlab CF direct solver, Matlab CG solver, CUSP CG solver, CUSP CG solver with Jacobi preconditioner, CUSP BICGS solver with preconditioner and CUSP CG solver. 49

4.3 Mean and Maximum percentage error comparison between CUSP CG solver and Matlab CG solver. One iteration of 5D-ODETLAP is run for measurement of errors. 53

5.1 Annual objectively analyzed climatology datasets on 33 standard depth levels on all 8 variables before compression. Each point uses 4 byte to store its value in single precision and thus total size of each dataset is 8.16 Mb. 54

5.2 Results of running 3D-ODETLAP with different sub-box sizes on part of Percent Oxygen Saturation dataset from surface to 4000 metres, which is a $128 \times 128 \times 32$ 3D matrix. Other parameters remain the same for all three tests. 55

5.3 Results of running 3D-ODETLATP on the 8 datasets derived from WOA 2005. Smooth variable R is set to 0.01, the size of a sub-box is $30 \times 30 \times 30$, initial point selection picks every 5 points along the regular grid, and refinement of point selection adds 20 points at each iteration and the size of the forbidden zone is 2. The size of each dataset is 7.42Mb with each point uses 4 bytes. For example, compressed Salinity dataset file size by 3D-ODETLAP is 60,037 bytes and thus has a compression ratio of 130:1. 56

5.4 Comprehensive compression performance comparison between 3D-ODETLAP and 3D-SPIHT. Due to the data size limitations of 3D-SPIHT, we extract eight $128 \times 128 \times 32$ 3D data matrices and apply both methods on them. Mean Percentage Errors are kept approximately the same for comparison purposes. The size of each dataset is 2Mb with each point uses 4 bytes. For example, compressed Salinity dataset file size by 3D-ODETLAP is 27,377 bytes and thus has a compression ratio of 77:1. 68

5.5 Monthly objectively analyzed climatology datasets [4,16,35] on 24 standard depth levels on four variables before compression. Each point uses 4 byte to store its value in single precision; thus total size of each 4D dataset is 71.20MB, which forms a 5D dataset 284.80MB totaling. . . . 69

5.6 Performance comparison between 5D-ODETLAP and 3D-SPIHT on the derived dataset, which is the same with the Data2 dataset in Table 3.2. The upper part of this table shows the results with approximately same mean percentage error of both methods; the lower part shows results with approximately same maximum percentage error. Uniformly random initial selection as in Section 3.2.5 is used for 5D-ODETLAP. The four values within brackets in the rows "Mean Err" and "Max Err" represent respectively values of each 4D dataset, including *apparent oxygen utilization, percentage oxygen saturation, dissolved oxygen* and *temperature*. Mean error is the average of all four 4D datasets' mean percentage error and similarly, Max error is the maximum of all four 4D datasets' maximum percentage error. Each point uses 4 bytes to store its value in single precision; thus total size of each dataset is 10.13MB. . 70

5.7 Data properties of WOA09_1 dataset, which is one of the eight test datasets. Each point uses 4 byte to store its value in single precision; thus total size of each 4D dataset is 17.80MB, which forms a 5D dataset 71.20MB totaling. 70

5.8 Top: Maximum percentage error of 5D-ODETLAP, JPEG 2000 and 3D-SPIHT with the same **mean** percentage error on eight different datasets. Bottom: The two column indicate the ratio of 5D-ODETLAP's compression ratio over the compression ratio of JPEG 2000 and 3D-SPIHT respectively. 71

5.9 Top: Mean percentage error of 5D-ODETLAP, JPEG 2000 and 3D-SPIHT with the same **maximum** percentage error on eight different datasets. Bottom: The two column indicate the ratio of 5D-ODETLAP's compression ratio over the compression ratio of JPEG 2000 and 3D-SPIHT respectively. 72

LIST OF FIGURES

1.1 $180 \times 360 \times 24$ Monthly Temperature (January) Data in *World Ocean Atlas 2005* (WOA 2005) 3

2.1 3D-SPIHT video coding system 7

2.2 ODETLAP approximation of nested squares 10

2.3 Computational power comparison between CPU and GPU [46]. 13

2.4 CUDA memory hierarchy [46]. 15

3.1 3D-ODETLAP algorithm outline 20

3.2 Forbidden Zone check: big red sphere is the forbidden zone of one point. the yellow points are inside this sphere and thus not included. All the blue points are outside the zone and included. 21

3.3 Illustration of solving time for original and transformed solver. The red line represents the solving time of original solver and the blue line represents that of transformed solver by normal equations 23

3.4 Reconstruction from 1% random selected points using weighted average method 24

3.5 The limited radius of impact during reconstruction from altering a single known point in the data. This plot shows the per-point difference between the 3D-ODETLAP solution for a $30 \times 30 \times 30$ 3D matrix with value equal to 1 and the ODETLAP solution for the same data with the central point's value modified to 1000. The difference in the solutions is restricted to a $15 \times 15 \times 15$ area around the altered point, with 88% of the affected points concentrated in the center $10 \times 10 \times 10$ area. . . . 37

3.6 Visual effect of different dividing strategies. Top left: Error of Simple Dividing Strategy; Bottom left: Error of Averaging Overlapping Area Strategy; Top right: Error of Weighted Averaging Strategy; Bottom right: Error of non-dividing Strategy 38

3.7 5D-ODETLAP framework outline. 39

3.8 4D Forbidden Zone check. The red sphere represents the forbidden zone of the red point in the center. The green point are abandoned. The blue ones are being considered. 40

3.9 2D illustration of *Four Dimension Dividing into Boxes* method. Reconstruction is done individually for each square in first run as marked by grey squares. The 50% overlapping blue squares also have computed value for each point from the second run, and we take a weighted average of both values in the overlapping area (blue) to produce a good approximation. 41

3.10 Outline of the further encoding part of the 5D-ODETLAP compression framework in Figure 3.7. 42

3.11 Binary position file compressed file size by 17 different coders. The uncompressed 5D binary position file is 2332800 bytes. 43

3.12 Floating point value vector final compressed file size by 11 different coders. 44

4.1 Outline of CUSP integration into Matlab. 45

4.2 A 4096×4096 square, symmetric and positive-definite linear system in 5D-ODETLAP. Number of non-zeros is 132608. 47

4.3 Performance comparison between CUDA solvers and Non-CUDA solvers in the 104976×104976 test dataset from Table 4.2. 51

4.4 3D illustration of compression results from Cholesky Factorization solver (Top Right) in Matlab and CUDA-based CG solver with Jacobi preconditioner(Bottom Right). The dataset is $16 \times 16 \times 16$ derived from the result of compressing a $16 \times 16 \times 16 \times 12 \times 4$ 5D dataset in WOA09. It represents one part of temperature data in Pacific Ocean in January, 2009. 52

5.1 Mean(left) and Max(right) percentage error as the number of points added at each iteration increases from 1 to 96. The dataset is $64 \times 64 \times 32$ 3D matrix derived from Percent Oxygen Saturation in WOA 2005. The number of selected points is 2100, smooth variable R is set to 0.01, subbox is $32 \times 32 \times 32$, initial point selection picks every 5 along the regular grid and the size of the forbidden zone is 3. 55

5.2 Maximum percentage error comparison between 3D-ODETLAP and 3D-SPIHT under same mean percentage error. 57

5.3 Compression ratio comparison between 3D-ODETLAP and 3D-SPIHT under same mean percentage error. 58

5.4 Rendering of partial Salinity dataset used in Table 5.4 59

5.5 Rendering of partial Percent Oxygen Saturation dataset used in Table 5.4. 60

5.6 Histogram of Salinity and Percentage Oxygen Saturation datasets used in Table 5.6 61

5.7 Number of points added to each 4D dataset within the 5D dataas, as the iterative process proceeds. The size of each 4D dataset is $48 \times 48 \times 24 \times 12$, and the size of the test 5D dataset is $48 \times 48 \times 24 \times 12 \times 4$. . . 62

5.8 Decrease of mean percentage error of each 4D dataset within the 5D dataset and the overall 5D dataset, which is the average mean percentage error of all 4D datasets. Test data is the same as that in Figure 5.7. 63

5.9 Decrease of maximum percentage error of each 4D dataset within the 5D dataset and the overall 5D dataset, which is the maximum percentage error of all 4D datasets. Test data is the same as that in Figure 5.7. . . 64

ACKNOWLEDGMENT

I would like to thank everyone who have helped me in my doctoral studies at Rensselaer Polytechnic Institute.

My first, and most earnest, acknowledgment must go to my advisor, Dr. W. Randolph Franklin, for bringing me into academic research, for his encouragement, guidance, and helping me reach my fullest potential. In every sense, none of this work would have been possible without him. I also wish to thank the many other people at RPI who have helped me along the way: Dr. Barbara Cutler, Dr. Peter Fox, Dr. Miriam Katz, Eddie Lau, Terry Hayten, Zhongyi Xie, Jake Stookey, and Chris Stuetzle. I would also like to thank committee members Dr. Mukkai S. Krishnamoorthy and Dr. Michael Wozny.

Lastly, I wish to thank my parents, Rongcong Li and Guangqiong Pan, for everything they have done for me.

ABSTRACT

This thesis describes HD-ODETLAP, a geospatial data compression technique to lossily compress high dimensional geospatial datasets. A five dimensional (5D) geospatial dataset consists of several multivariable 4D datasets, which are sequences of time-varying volumetric 3D geographical datasets. These datasets are typically very large in size and demand a great amount of resources for storage and transmission. HD-ODETLAP consists of work from two steps.

Firstly, we build the foundation of HD-ODETLAP method from 3D-ODETLAP method, which targets at compressing 3D geospatial datasets. With proper point selection, our 3D-ODETLAP method approximates uncompressed 3D data using an over-determined system of linear equations. Then this approximation is refined via an error metric. These two steps work alternatively until a predefined satisfying approximation is found. This chosen representative sample set of orginal 3D dataset is then encoded using simple Run Length Encoding (RLE) and prefix coding technique.

Secondly, based on 3D-ODETLAP, we present 5D-ODETLAP, a lossy compression algorithm and implementation for high dimensional geospatial data. 5D-ODETLAP exploits the spatial dependency and autocorrelation in every dimension in these large datasets. This is an advance on traditional methods that compress only lower dimensional slices. 5D-ODETLAP greedily selects a characteristic subset of the original 5D dataset, chosen to minimize information loss. The selected set of points is further compressed using a coder built from classic encoding methods. That coded set of points is the compressed representation of our dataset. To uncompress the data, 5D-ODETLAP recomputes the values at each point in 5D by solving a sparse overdetermined linear system of equations.

After preliminary test of 5D-ODETLAP, we optimize it by using a much more advanced encoding method than the simple RLE and prefix coding. The second

advance in 5D-ODETLAP is our incorporation of a CUDA-based conjugate gradient linear solver into this framework. That exploits the massive, and inexpensive, parallelism available in modern GPUs. We have interfaced CUDA with Matlab to maximize programming efficiency and to minimize data transfer overhead. We have tested 5D-ODETLAP with various datasets and error metrics. With the same mean percentage error, compressed file size by 5D-ODETLAP is 7.67 and 2.14 times as small as that by JPEG2000 and 3D-SPIHT respectively in our eight test datasets on average. 5D-ODETLAP's advantage is even larger under the same maximum percentage error. 5D-ODETLAP has no restrictions in the data types, and it has the flexibility to properly adjust the parameter setting for other datasets with spatial and temporal redundancy.

1. INTRODUCTION

1.1 Motivation

The collection and dissemination of high dimensional geospatial data is among the fastest areas of growth, due to advances in data-acquiring technology and an increasing realization of the benefits of such data in commercial usage and environmental assessment. Government and defense organizations have made great efforts to make this data publicly available for research, by preserving a numerical record of our global atmospheric and geological environment. The National Oceanic and Atmospheric Administration (NOAA) and its subsidiary organizations, including the National Oceanographic Data Center (NODC) and the National Geophysical Data Center (NGDC), play roles as national repositories and dissemination facilities for global geospatial data. They provide a record of earth's changing environment, and support numerous research and operational applications. For example, four dimensional (4D) or five dimensional (5D) data are often collected from ocean studies. These data include *temperature, salinity, nitrate, oxygen saturation*, and *silicate* of the sea water [4, 5, 15, 16, 17, 18, 35, 36].

A single point in a 5D dataset has a coordinate *(x, y, z, t, p)* and a scalar value associated with it. The *x, y, z* are the 3D spatial coordinates, while t is the temporal coordinate. The p value represents different variables, such as *temperature, salinity* and *oxygen utilization*. So this means a 5D dataset consists of several multivariable 4D datasets, which keeps records of different environmental variables in the same area at a certain period of time. Each 4D geospatial dataset describes temporal changes of spatial geological variables, such as *temperature* of ocean water, in turn as a sequence of three-dimensional (3D) volumes. Similarly, a 3D geospatial dataset is a sequence of two dimensional(2D) datasets in the same 2D area. As the dimension goes higher, the amount of data will consequently increase exponentially, which makes storage and transmission of these data more difficult than ever before even at today's internet speed. Without effective ways of utilizing these data, it's a waste of time and effort to collect and store them.

Current improvements on data storage and communication methods are reducing the cost of storing and transmitting large amounts of geospatial data. Nevertheless, as research progresses, the decreasing cost of storage and bandwidth will not keep pace with the rate of growth in high dimensional geospatial data. Furthermore, the emergence of real-time rendering of 3D earth environment such, as Google Earth, makes high dimensional geospatial data compression a significant area of research. The data compression problem also limits the application of innovative three dimensional map algebra operations [42] and many other similar operations on high dimensional geospatial data. Some research work has been done in using hierarchical spatial data structure in GIS to offer advantages of data compression [20]; Also, spatial dependency and spatial autocorrelation research could also provide insights into lossy data compression [19]. Unfortunately, there is not much published effort in this field, and work still needs to be done.Therefore, it's still a challenge to design a progressive and effective compression scheme for efficient storage and transmission of rising high dimensional geospatial data.

Advances in physical storage media and transmission technology have been constantly reducing the cost of effectively using high dimensional geospatial data. However, those achievements lag behind the growth in the datasets' size due to improvements in surveying technology. Moreover, there is an increasing interest both in academic and commercial applications, such as Google Earth, of real-time rendering of the 3D earth environment. These facts make a dedicated high dimensional geospatial data compression method important. Some research work has been done in using a hierarchical spatial data structure in GIS for data compression [20]. Data compression can be defined as the removal of the redundancy that a dataset inherently possesses. Spatial dependency and autocorrelation research could provide insights into lossy data compression [19]. But the current research in geospatial data compression has not yet focused on high dimensional data autocorrelation. They ignore the data dependency in those high dimensions, and so do not obtain a high compression ratio. Therefore, it's still a challenge to design a progressive and effective compression scheme for efficient storage and transmission of rising high dimensional geospatial data.

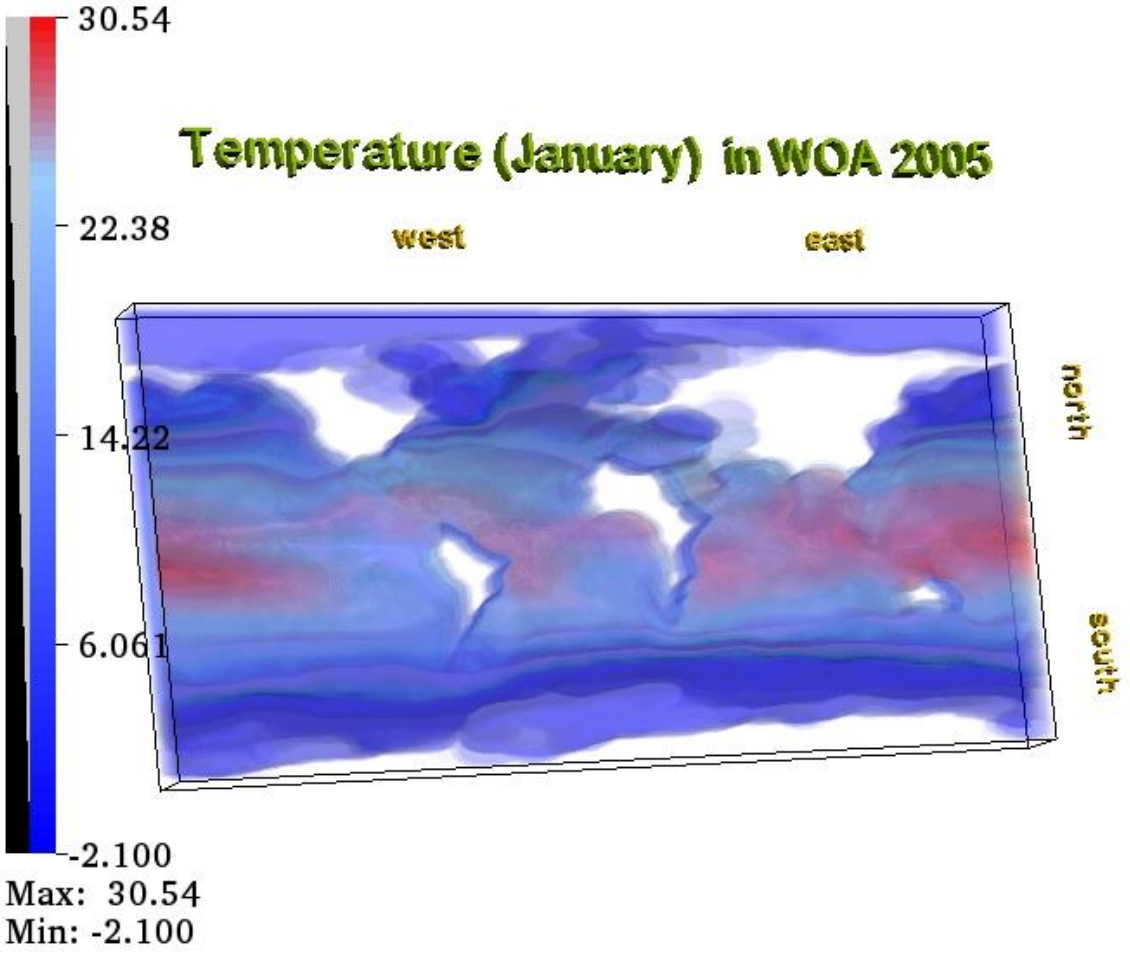

Figure 1.1: $180 \times 360 \times 24$ **Monthly Temperature (January) Data in *World Ocean Atlas 2005* (WOA 2005)**

For example, a $180 \times 360 \times 24 \times 12 \times 4$ 5D dataset in *World Ocean Atlas 2009* (WOA 2009) [5, 17, 18, 36] contains the scalar value of *temperature, salinity, nitrate*, and *silicate* in a 12 month period at 24 standard depth levels of the global ocean. This dataset is stored as single precision floating points and compressed using gzip, which is a general purpose text compression method. This dataset is 284.77 Mb, and note that this is a very coarse dataset (1 degree). Figure 1.1 visualizes January's data derived from the monthly temperature data in WOA 2005.

1.2 HD-ODETLAP

To begin with, we use the 3D Over-determined Laplacian Partial Differential Equation method (3D-ODETLAP) to approximate and lossily compress 3D marine data. 3D-ODETLAP is an extension of, but essentially different from, the Laplacian Partial Differential Equation. The essential difference between them is that the Laplacian PDE only assigns one average equation to every data point, while 3D-ODETLAP assigns one exact equation to every known data point besides the average equation. So for all known data points, their value is equal to two different values, which makes 3D-ODETLAP an over-determined linear system. The least square solution to this system is biased at certain data points if we choose different weights over the equations of those points. But in the Laplacian PDE, the value of each data point is only assigned to one average equation, and thus the PDE only has one exact solution. So this means 3D-ODETLAP has completely different mathematical properties from the Laplacian PDE.

Marine data includes single-valued salinity, temperature, oxygen and phosphate, etc. First we construct an over-determined system using random point selection; then we use an over-determined PDE to solve for a smooth approximation. The initial approximation might be very coarse due to the limited number of selected points. Furthermore, we refine this approximation with respect to the original marine data by adding points with largest error, and then running 3D-ODETLAP again on the augmented representation to gain a better approximation. These two steps work alternately until a stopping criteria is reached, which is usually the maximum error. The selected sample point set is then encoded using *Run Length Encoding* (RLE) and prefix coding technique to obtain a final compressed file. We also analyze the size versus accuracy tradeoff. Meanwhile, there are many missing values in marine data due to physical difficulties in measurement. Our 3D-ODETLAP can interpolate those missing values and produce a smooth-looking image without visible artifacts.

The above 3D-ODETLAP method builds a solid foundation for HD-ODETLAP to compress higher dimensional data. For 5D geospatial data compression, we introduce the 5D Over-determined Laplace Partial Differential Equation (5D-ODETLAP)

method, which is the first to exploit correlations across high dimensions simultaneously. This compression method has two stages: high dimensional data decorrelation, and further encoding. In the data decorrelation stage, we first construct overdetermined systems for each 4D slice using a designated point selection method. Then we solve these systems for a smooth 5D approximation using a conjugate gradient linear solver with Jacobi preconditioner based on the Compute Unified Device Architecture (CUDA) [46]. This approximation has limited information since the initial number of selected points is small. But we then iteratively improve this approximation by adding more points, thus more information, into our selected sample point set in 5D space. Our point-adding strategy is to select those points with largest errors of the 5D approximation compared to the original 5D dataset. These two steps work alternately until we reach a stopping criterion. Once we have a representative sample of points for the original 5D dataset, we then store these points using a prescribed number of bits in the further encoding stage. We split the information of sample point set into 5D coordinates and a scalar value vector. Then we compress them separately using traditional data quantization and entropy encoding methods tailored to meet the needs of a very sparse sample point set.

Since 4D data compression is actually part of the 5D-ODETLAP method, HD-ODETLAP method provides a framework of compressing high dimensional geospatial data in 3D, 4D and 5D.

1.3 Thesis Outline

This thesis is organized as follows. In Section 2, we briefly walk through current general compression techniques followed by a short introduction of original ODETLAP method. Then we give a short background on current research about linear solution in CUDA. In Section 3, we demonstrate the details about HD-ODETLAP with algorithm outline and implementation techniques both in 3D-ODETLAP and 5D-ODETLAP. Section 4 describes our research in CUDA-based linear solver for HD-ODETLAP. In Section 5, we first present the result of 3D-ODETLAP and then the preliminary result of 5D-ODTELAP. Then we demonstrate the results of the optimized 5D-ODETLAP with linear solver and better further encoding scheme.

Results comparison between HD-ODETLAP, JPEG 2000 and 3D-SPIHT are also shown in this section. Finally in Section 6 we conclude the whole thesis and discuss briefly our current ideas for future work in Section 7.

2. Background

2.1 General Compression Methods

There are two types of data compression techniques: lossless and lossy. The lossless schemes preserve all the information in the dataset, allowing exact reconstruction of the original data. But they usually suffer from a relative low compression ratio when compared to lossy schemes. In contrast, lossy compression methods allow deleting some of the information to achieve a much higher compression ratio. Some data types, such as 3D medical images, require lossless storage since a false diagnosis could cause legal difficulties. But this is usually not the case with geospatial data. The data analysis, modeling and visualization are not affected as long as the error is controlled.

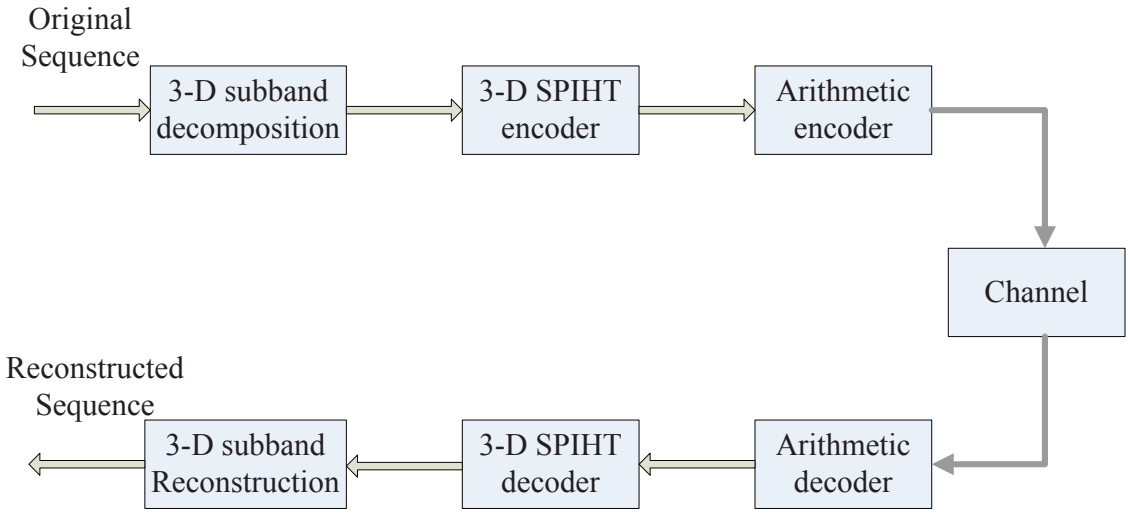

Figure 2.1: 3D-SPIHT video coding system

Various compression methods have been created for lower dimensional data, such as 2D images [53, 55, 57, 60], 3D image sequences [28, 41, 59], videos [49] and 4D Functional magnetic resonance imaging (FMRI) [30]. Among those methods, the wavelet-based lossy and lossless ones are the most popular. For example, JPEG 2000 [57] uses irreversible and reversible wavelet transforms for its lossy and lossless compression for 2D images. Other compression schemes are also available for 2D

image compression, including the widely used Tagged Image File Format (TIFF) [1]. It contains several different compression methods, among which the CCITT (International Telegraph and Telephone Consultative Committee) Group 3 and Group 4 are specially designed for encoding 1-bit image data. For 3D video data, Three Dimensional Set Partitioning in Hierarchical Trees (3D-SPIHT) [27] also bases its compression scheme on the 3D wavelet transform shown in Figure 2.1, with which we will compare our method later in this thesis.

For 4D data, including video and time-varying 3D geospatial volumes, there are specially designed methods, such as 4D wavelets [64], run length encoding (RLE) [3] and discrete cosine transform (DCT) [38]. 3D compression methods can also be applied to 4D data, since 4D data could be ghandled as a sequence of 3D volumes. But methods that exploit the temporal autocorrelation between volumes usually outperform their 3D counterparts, such as the video compression methods using motion compensation technique [50] and 4D-SPIHT [66]. These methods fully utilize the spatial and temporal data redundancy in all the dimensions; thus they can achieve a higher compression ratio.

Various compression schemes have been proposed for 2D, 3D and 4D gridded data. Most of those schemes are focused on 2D images [53, 57, 60] and 3D image sequence data, especially in multidimensional medical images [28, 41]. These 3D compression methods either compress image slices independently (ignoring the correlation in the third dimension), or compress the whole 3D volume using 3D wavelets transform, such as 3D-SPIHT [27] shown in Figure 2.1, with which we will compare our method later in this thesis. In the former case, JPEG2000 [53] and JPEG-LS [60] are the most popular ones.

The above existing popular compression methods all consist of several coding steps. For example, JPEG 2000 has five stages: color components transformation, tiling, wavelet transform, quantization and arithmetic coding. These are many different coding methods, including RLE [3], Arithmetic encoding [61], Huffman coding [29], Prediction by Partial Matching [43] and Dictionary-based coding [9]. These methods have their pros and cons on datasets with different internal structures. Researchers can build their own compression methods based on all these

existing coders to achieve the best compression result.

In geography, there is an increasing awareness of the importance of compression and progressive transmission of spatial data. [48] uses adaptive run-time data compression on the spatial data generated from hyperspectral imaging applications. [26] systematically studied different compression schemes for efficient maintenance and dissemination of spatial databases. Research on compression and spatial decorrelation has been done in digital terrain models [10]. HD-ODETLAP doesn't explicitly use decorrelation schemes in compression, but it does so effectively via greedy sampling and reconstruction.

2.2 ODETLAP

2.2.1 ODETLAP Framework

Introduced by Franklin, the original ODETLAP [13, 14] is designed for terrain compression and reconstruction [31], and HD-ODETLAP boldly extends this idea to high dimensional geospatial data compression. It takes a sparse set of known elevations in a Digital Elevation Model (DEM) as input, and produces a complete elevation map. As shown in Figure 2.2, ODETLAP approximates the input data and smooths the sharp edges. This nice feature can be used to approximate and reconstruct the DEM. For this purpose, ODETLAP builds up a linear system with multiple linear equations. These equations have the following constraints. First, for all points, known and unknown, the elevation value equals the average of its four neighbors, as expressed by:

$$u_{i,j} = (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1})/4 \tag{2.1}$$

Second, for all known points, the elevation is equal to the known elevation at that point:

$$u_{i,j} = h_{i,j} \tag{2.2}$$

These equations form a large over-determined system of linear equations. Equation 2.1 enforces smoothness between adjacent points. But Equation 2.2 can overcome this constraint by introducing a new set of equations. For example, adjacent points

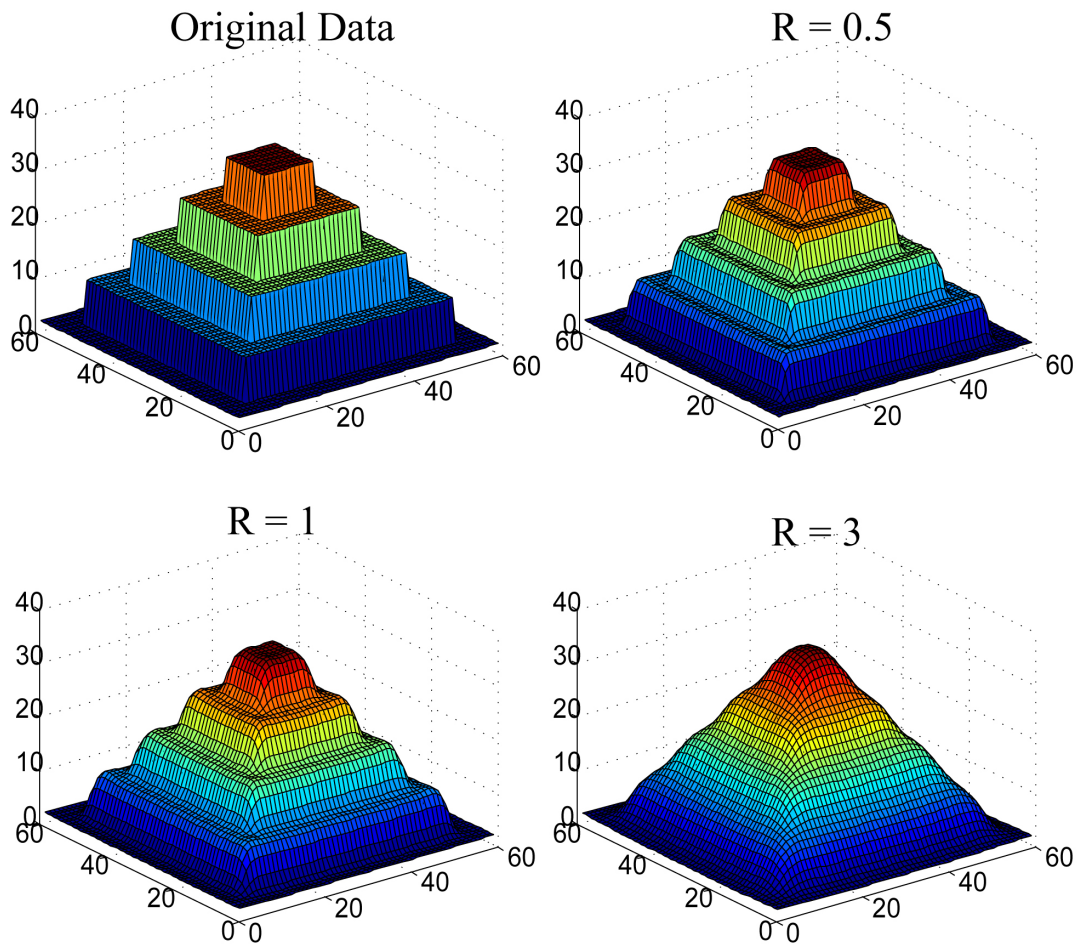

Figure 2.2: ODETLAP approximation of nested squares

with known elevations may not be equal to the average of the neighbors. For this reason, there is no exact solution, and an approximation is made in the least squares solution. An additional parameter R is added to trade off between smoothness and accuracy of the known points. Mathematically, we multiply both sides of Equation 2.1 with this parameter R and keep Equation 2.2 unchanged in order to weight Equation 2.1 over Equation 2.2 in its least squares solution. A small R will approximate a determined solution, and the solution will be more accurate in terms of the known elevations. A very large R weights Equation 2.1 more and thus will result in a smoother solution, but some of the known elevations may change. Xie [62] and Gousie [21, 22] have a more in-depth discussion of ODETLAP in 2D terrain compression.

2.2.2 ODETLAP Slope Compression

Dan Tracy [58, 63] further extended the original ODETLAP by adding two equations

$$u_{i+1,j} - u_{i-1,j} = h_{i+1,j} - h_{i-1,j} \tag{2.3}$$

$$u_{i,j+1} - u_{i,j-1} = h_{i,j+1} - h_{i,j-1} \tag{2.4}$$

where the variables u and h represent the same meaning as in Equations 2.1 and 2.2. These two slope equations are structured in this manner to match the way quantities are used in the Zevenbergen-Thorne slope method [65], which is used for measurement of slope error. So the modified over-determined system contains thress sets of equations:

- $R \times (u_{i,j} = (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1})/4)$
- $u_{i,j} = h_{i,j}$
- $RS \times (u_{i+1,j} - u_{i-1,j} = h_{i+1,j} - h_{i-1,j})$ and $RS \times (u_{i,j+1} - u_{i,j-1} = h_{i,j+1} - h_{i,j-1})$

where parameter R remains the same and RS is the weight of the two slope equations. With a slightly modified compression process from the original ODETLAP method, ODETLAP slope compression can compress the DEM data while minimizing slope error and Root Mean Square (RMS) error.

2.2.3 Parallel ODETLAP

Jake Stookey [54] introduced a new parallel ODETLAP for large scale DEMs compression. The basic idea of this parallel algorithm is to divide the datasets into patches, run ODETLAP on each patch, and then merges the patches together.

This method gives two distinct improvements on speed. First, it reduces the complexity through dividing the DEM into patchs, which is similar to the *Four Dimension Dividing into Boxes* in Section 3.2.4. By this way, the execution time grows almost linearly with the size of the input, even when running on a single processor. On the other hand, the calculation of ODETLAP on the patches can be done concurrently in a parallel or distributed environment. In experiments, the parallel ODETLAP implementation takes 2 seconds to run ODETLAP on a 800×800

elevation map using 128 processors, while the original version of ODETLAP takes nearly 10 minutes on a single processor. Furthermore, this algorithm allows users to process large datasets of size 16000×16000 on a cluster of computers, which take the original ODETLAP prohibitively long to run.

2.3 Linear Solution in CUDA

2.3.1 GPGPU background

General-Purpose computation on Graphics Processing Units (GPGPU) [37] is the technique used to perform general purpose computation using non-graphics data on GPUs in order to take advantage of the enormous computing power of stream processors. Traditionally, GPU functionality has, since it was invented, been very limited and has only been used to accelerate the computation on graphics pipeline. And the programming interfaces such as OpenCL [44] are relatively difficult to learn and work for entry-level developers. Being aware of those limitations and the significant waste of computing power on hardware, GPGPU technique makes it possible for developers to utilize the growing computing power on GPU hardware in a relatively easy and understandable way. As the producers of cutting-edge GPU hardware, NVIDIA and AMD both published their GPGPU programming architecture. AMD's ATI Stream technology [2] kicked off the GPGPU revolution with the introduction of CTM(Close To Metal) [23] in 2006. NVIDIA also introduced CUDA in November 2006. From 2006 to present, both ATI Stream, now AMD Accelerated Parallel Processing (APP), and CUDA [46] gained a lot of attention in the computing field, and a large number of active developers in industry and academia have been constantly using and supporting them.

2.3.2 CUDA programming architecture

GPU has evolved into a highly parallel, multi-threaded, many core processor with tremendous computational horsepower and very high memory bandwidth. It has outperformed its CPU counterparts for all these years as illustrated in Figure 2.3.

Within the CUDA programming model, there is a term called "kernel", which

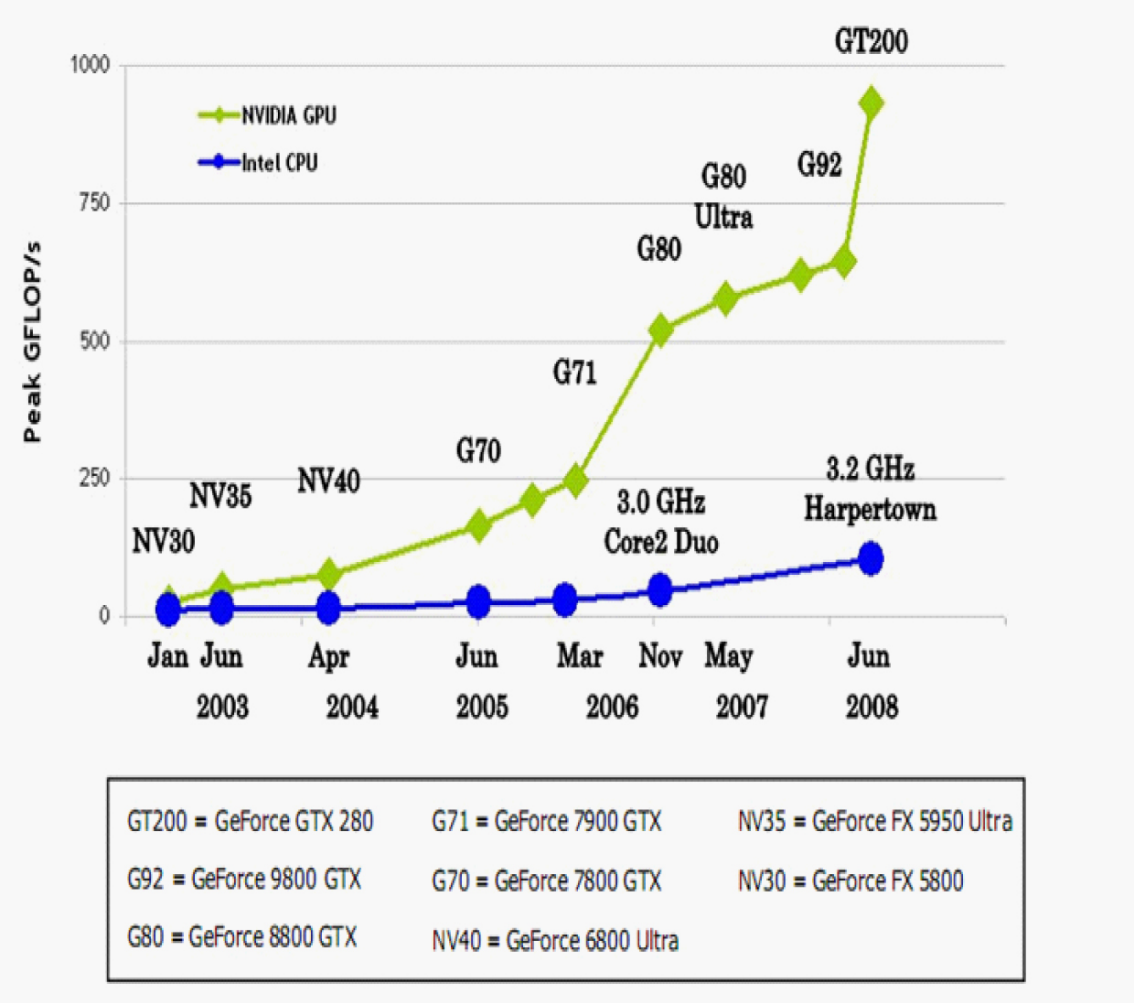

Figure 2.3: Computational power comparison between CPU and GPU [46].

actually means the program running in parallel on the device (NVIDIA CUDA-enabled GPU). The kernel is nothing but a subroutine to enable data-parallel computation. A developer specifies a certain number of primitive threads which will simultaneously execute the kernel program. Threads are organized into "thread block" and the number of threads at each block is chosen by the developer according to hardware specification. This number is limited to 1024 per block in Nvidia GPUs with 2.x compute capability. All threads within a block can "talk" to each other effectively using "shared memory" in this block. They can also synchronize with one another using kernel API.

A grid is a batch of blocks with the same thread hierarchy, which executes the

kernel simultaneously. Grid is mainly for dealing with large amounts of data, but blocks can only communicate with each other via global memory, which is ineffective and thus can not synchronize with one another. The CUDA memory hierarchy is shown in Figure 2.4.

CUDA contains several levels of memory including local memory, shared memory and global memory, which has access time and capacity from lowest to highest. They are respectively in the level of thread, block and grid and thus have different characteristics. Constant memory and texture memory are cached global memory with read-only features. The core of acceleration of a CUDA program lies in the appropriate usage of different types of memory.

2.3.3 Motivation

The need for fast sparse linear solvers arises in various computational disciplines, and accordingly, efficient implementations of these solvers are often among the core research areas in many institutions. One type of the most common linear systems is sparse, symmetric and positive-definite. It's often constructed when numerically solving partial differential equations (PDE) in mesh smoothing, fluid simulation and solid mechanics. In these settings a Conjugate Gradient (CG) solver (or one of its mathematical variants) is often appropriate. CG is the most popular iterative method for solving large systems of linear equations. CG is effective for systems of the form

$$Ax = b \tag{2.5}$$

where x is an unknown vector storing the solution, b is a known vector, and A is is a known, square, symmetric, positive-definite (or positive-indefinite) matrix. Solving this system usually accounts for the dominant cost in many applications.

As a solution, high performance graphics processing units (GPUs) manufacturers, such as NVIDIA and AMD, introduced a developer-friendly programming interface for their floating point hardware, which have the hardware computing power to outperform CPUs by an order of magnitude [25]. So GPUs are strong contenders as powerful computational engines for intensive floating point applications suitable for parallelism. With their existing utility in graphics-related computing,

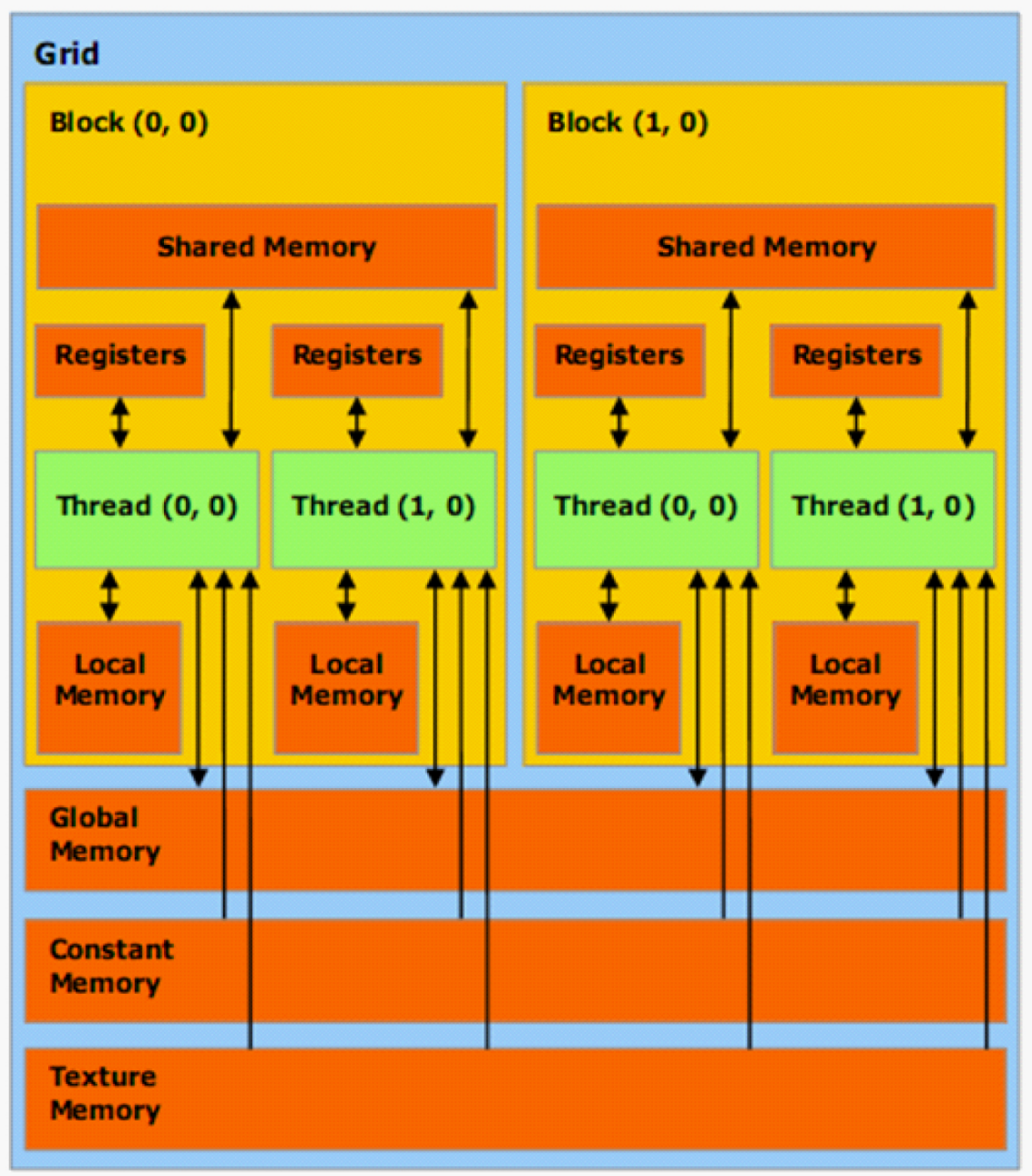

Figure 2.4: CUDA memory hierarchy [46].

GPUs also gain commercial success as a class of future computing architecture and next-generation super-computing [47]. Active development of GPU-based linear solvers [8, 11] has shown great speed-up using current GPUs in solving different linear systems. Especially in the field of computer graphics, GPU-based solver has been used to achieve fast texturization and visualization. For example, [24] used a multigrid solver on GPU to solve the Navier-Stokes equations in order to generate marbling textures for 3D models. These solvers are often available only in popular and fundamental languages such as C/C++ for the purpose of easy usage and high efficiency.

These languages are often not designed for matrix manipulation, and researchers usually use more advanced languages, such as Matlab and R, for applications involving heavy matrix operations and solving linear systems. Nevertheless, these advanced languages have not yet developed their corresponding linear solvers based on GPU, and the performance of existing CPU-based solvers are inherently inefficient. As an emerging technology, GPU-based linear solvers are just not yet available in these mature, or commercialized in some cases, languages. Thus, a solution to incorporate these GPU-based solvers into advanced languages is urgently needed.

In Chapter 4, we explore the technique for integrating and CUDA-based solvers and high-level language such as Matlab in order to provide an overhead-free solution to GPU-based linear solvers. Though implemented in C++ and CUDA, this integrated Matlab linear solver does not require users to write CUDA code, which has a relatively sharp learning curve compared to other languages. This solver is available as a simple Matlab function call and thus does not introduce any data transfer overhead between Matlab and the solver program. This feature is particularly important when the system is large and sparse. On the other hand, to meet the needs of our 5D-ODETLAP method, we use the linear system constructed from 5D-ODETLAP as a test system to measure the solving time and residual of solutions. After comprehensive tests, we choose the appropriate preconditioner, linear solver and size of the residual of consequent solution. Furthermore, since 5D-ODETLAP is a lossy compression method in nature, only a relatively coarse solution to our

system is needed. Thus the solving time is greatly shortened compared with our original CF direct solver in Matlab.

3. HD-ODETLAP

The proposed HD-ODETLAP high dimensional geospatial data compression framework consists of the 3D-ODETLAP method and the 5D-ODETLAP method. 3D-ODETLAP targets at compressing three-dimensional datasets, while 5D-ODETLAP extends it to compressing 4D and 5D datasets.

3.1 3D-ODETLAP

3.1.1 Definition

As implied by the name, 3D-ODETLAP, or Three Dimensional Over-Determined Laplacian Partial Differential Equation, is an extension of a Laplacian PDE $\frac{\delta^2 z}{\delta x^2} + \frac{\delta^2 z}{\delta y^2} = 0$ to an overdetermined system of equations [54, 62]. Each unknown point induces an equation setting it to the average of its 3, 4, 5 or 6 neighbors in three dimensional space. We have the equation

$$u_{i,j,k} = (u_{i-1,j,k} + u_{i+1,j,k} + u_{i,j-1,k} + u_{i,j+1,k} + u_{i,j,k-1} + u_{i,j,k+1})/6 \tag{3.1}$$

for every unknown non-border point, which is equivalent to saying the volume satisfies 3D Laplacian PDE,

$$\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} + \frac{\delta^2 u}{\delta z^2} = 0 \tag{3.2}$$

This equation has the following limitations:

- The solution to this Laplace equation never has a relative maximum or minimum in the interior of the solution domain, this is called the maximum principle [52] so local maxima are never generated.
- Different marine data may have different data distributions. So the exact solution to the Laplacian PDE doesn't have the mathematical flexibility to approximate different marine data.

To avoid these limitations, an over-determined version of the Laplacian equation is defined as follows: apply the equation 3.1 to every non-border point, both

known and unknown, and a new equation is added for a set S of known points:

$$u_{i,j,k} = h_{i,j,k} \tag{3.3}$$

where $h_{i,j,k}$ stands for the known value of points in S and $u_{i,j,k}$is the "computed" value for every point, like in equation 3.1. Because the number of equations exceeds the number of unknown variables, this means the system is over-determined. Since the system is very likely to be inconsistent, instead of solving it for an exact solution (which is now impossible), an approximate solution is obtained by minimizing the error. Equation 3.1 is satisfied for each point, making it the average of its neighbors, which makes the generated surface smooth and resembles the real world situation. However, since we have known points where equation 3.3 is valid, they are not necessarily equal to the average of their neighbors. This is especially true when we have adjacent known points. Therefore, for points with multiple equations we can choose the relative importance of accuracy versus smoothness by adding a smoothness parameter when solving the over-determined system [12]. In our implementation, equation 3.1 is weighted by R relative to equation 3.3 which defines the known locations. So a very small R will approximate a determined solution and the surface will be more accurate while a very large R will produce a surface with no divergence, effectively ignoring the known points. Instead of interpolation, approximation is a more suitable term for this method because the reconstructed volume is not guaranteed to go through the input data points. 3D-ODETLAP can be used as a lossy compression technique since the original terrain can be approximated with some error using the set of points S for equations 3.1 and 3.3.

3.1.2 Algorithm Outline

The 3D-ODETLAP algorithm's outline is shown in Figure 3.1 and the pseudo code is given below. Starting with the original marine volume data matrix there are two point selection phases: firstly, the initial point set S is built by a simple regular grid selection scheme and a first approximation is computed using the equations 3.1 and 3.3. Given the reconstructed surface, a stopping condition based on an error measure is tested. In practice, we have used the mean error as the stopping

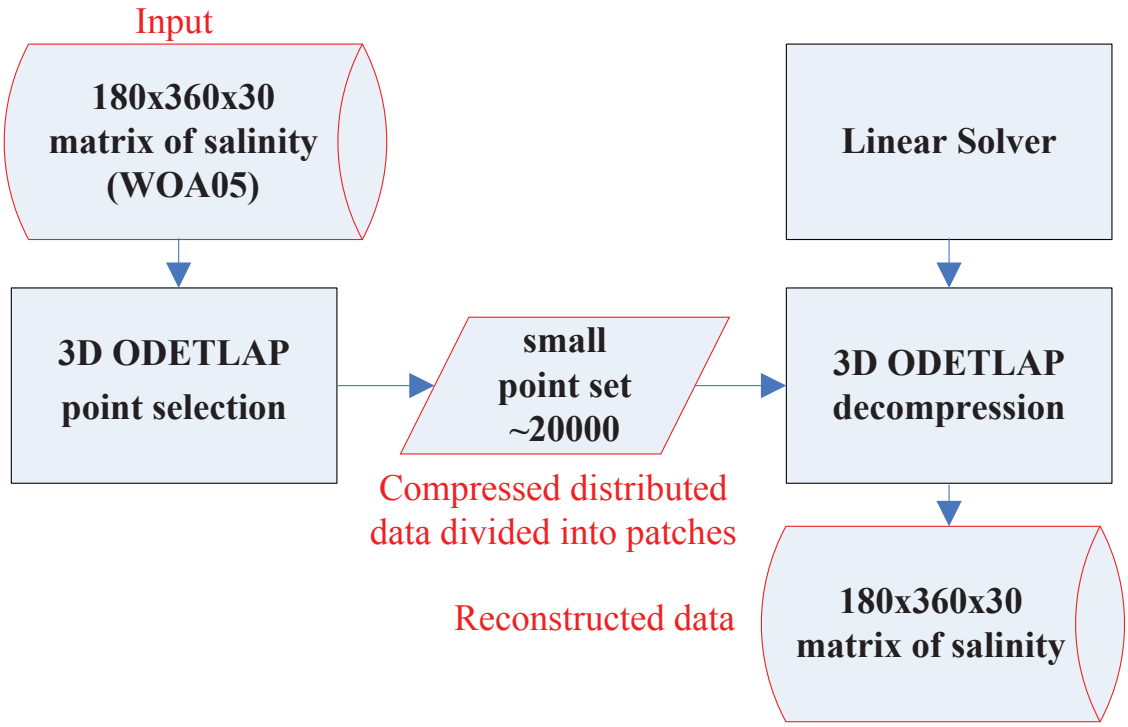

Figure 3.1: 3D-ODETLAP algorithm outline

condition. If this condition is not satisfied, the second step is executed. In this step, k≥1 points with biggest error are selected based on our "forbidden zone" method described below and then added into the existing point set S; this extended set is used by 3D-ODETLAP to compute a more refined approximation. As the algorithm proceeds, the total size of point set S increases and the total error converges.

Input: $3D - MarineData : V$

Output: $PointSet : S$

$S = RegGridSelection(V)$

$Reconstructed = 3D - ODETLAP(S)$

while $MeanError > Max_MeanError$ **do**

  $S = S \cup Refine(V, Reconstructed)$

  $Reconstructed = 3D - ODETLAP(V)$

end

return S

Algorithm 1: 3D-ODETLAP algorithm pseudo code

3.1.3 Forbidden Zone

The refined point selection strategy has flaws in that those selected points are often clustered due to high value fluctuation within a small region. In oceanographic datasets, if one point with large error is far away from others, it is most likely that its adjacent points are also erroneous and will be selected as well. Therefore, refined points selected may be redundant in some regions, which is a waste in compression.

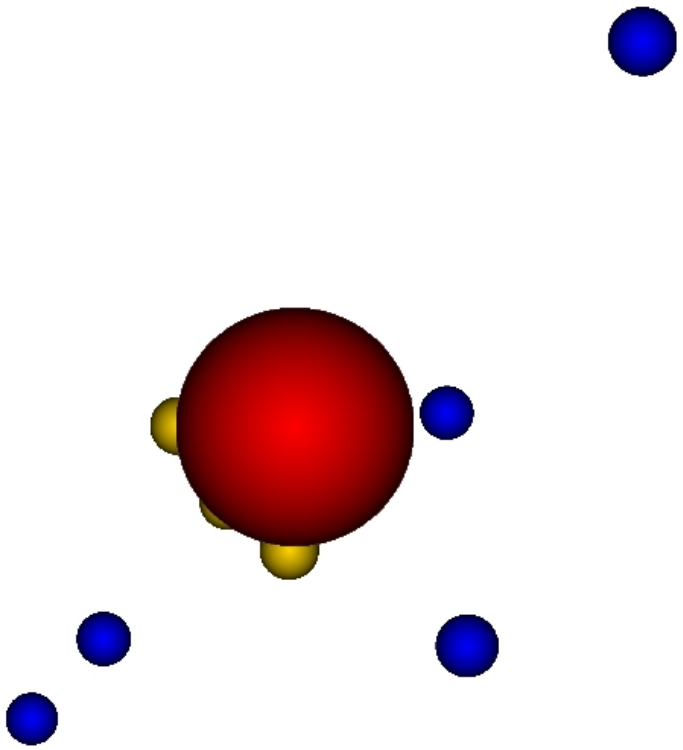

Figure 3.2: Forbidden Zone check: big red sphere is the forbidden zone of one point. the yellow points are inside this sphere and thus not included. All the blue points are outside the zone and included.

Forbidden zone is the check process we use to add new refined points: the spatial local neighbor of the new point will be checked to see if there is any existing

refined points added in the same iteration. If yes, then this new point is abandoned and the point with next biggest error is tested until we have a predefined number of refined points. Details are shown in Figure 3.2.

3.1.4 Implementation Speed-up

3.1.4.1 Normal Equation

In order to solve the linear system $Ax = b$ when A is asymmetric, we can solve the equivalent system

$$A^T Ax = A^T b \tag{3.4}$$

This is Symmetric Positive Definite in our case because in our *over-determined* system, A is a rectangular matrix of size $n \times m$, $m < n$. This system is known as the system of the *normal equations* associated with the least-squares problem,

$$minimize||b - Ax||^2 \tag{3.5}$$

Before applying *normal equations* method on our *over-determined sparse linear system*, our underlying solver to solve the over-determined system uses sparse QR decomposition in Matlab. It runs much slower than the Cholesky factorization (CF) [39], which solves the Symmetric Positive Definite linear system as shown in figure 3.3. Even with the overhead introduced by matrix multiplication, the *normal equations* method still solves our linear system significantly faster. Please note that the *normal equations* method enables us to solve small linear systems very quickly, which makes our next scheme *Dividing into Boxes* feasible.

3.1.4.2 Dividing into Boxes

As you can see from Figure 3.4, solving a reasonable large 3D matrix, i.e., $100 \times 100 \times 100$, still takes too much time. Linear regression tests show that 3D-ODETLAP is running approximately at $T = \Theta(n^6)$ for an $n \times n \times n$ matrix. And the memory cost is also high: solving a $90 \times 90 \times 90$ 3D dataset takes about 9.8 CPU-hours and 55 GB of writable memory on a four 2.4GHZ processor workstation with 60 GB of main memory running Ubuntu 9.04 and 64-bit Matlab R2009a. This

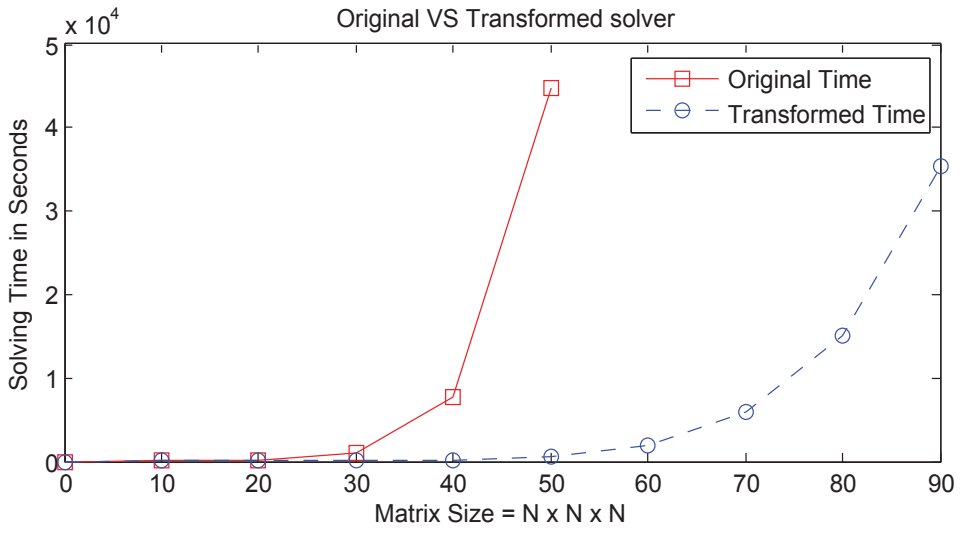

Figure 3.3: Illustration of solving time for original and transformed solver. The red line represents the solving time of original solver and the blue line represents that of transformed solver by normal equations

is unacceptable because the 3d datasets in the real world are often significantly larger than the test one, and the running time may be prohibitively long.

Within datasets, some points are quite distant and thus have nearly no influence on each other. Figure 3.5 shows that when we assign a high value to one point in the input to 3D-ODETLAP, only those points within a small neighborhood of this point are affected. Beyond that small region, the effect becomes negligible. This supports our hypothesis that it should be possible to divide a large dataset into separate boxes, run 3D-ODETLAP on them individually, and achieve similar results to the non-box 3D-ODETLAP solution. Specifically, we created a $64 \times 64 \times 64$ 3D matrix A based on function $A = (max_distance - distance)^2$, where $max_distance$ is the maximum distance from any points within the cube to its center and distance is the distance of this point to the center of the cube. Then we divide compressed A into a 4×4 matrix whose elements are $16 \times 16 \times 16$ boxes. Then we run 3D-ODETLAP on each box individually and merge them together as shown in Figure 3.4.

Nevertheless, if we simply merge all the boxes together, we tend to have large errors at the edge and plane of a box, where there is less information to work with than running 3D-ODETLAP on the matrix as a whole.

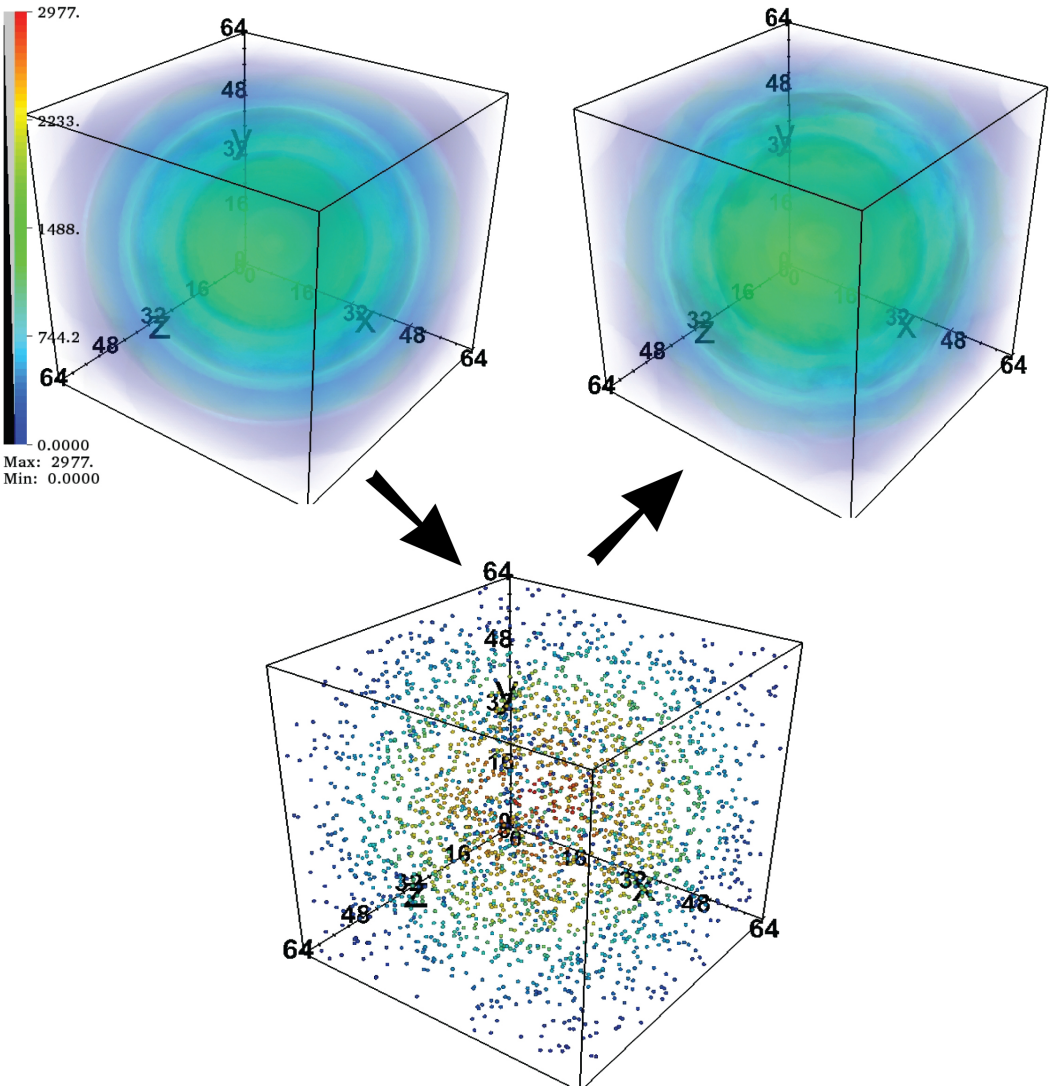

Figure 3.4: Reconstruction from 1% random selected points using weighted average method

Our solution is as follows: run 3D-ODETLAP twice on these data. First, 3D-ODETLAP is run individually on each box. Then, 3D-ODETLAP is run on the inner $48 \times 48 \times 48$ matrix which is the inner part of matrix A with each box size of $16 \times 16 \times 16$. These two runs enable us to have two calculated values for each point in the inner $48 \times 48 \times 48$ matrix and only one value for the rest of matrix A. Simply taking the average of both calculated values will reduce the errors at the edges and planes to some extent as shown in Figure 3.6 bottom left. But since we run 3D-ODETLAP redundantly on each point, we have the option to bias those

erroneous edge and plane points' values and select those values close to the center of another box.

In order to ignore those values of edges and plane points, we use a *Euclidean Distance* to assign weight to the two values of each point. The closer the point is to the center of its box, the more weight its value has and vice versa. For example, if the point has a distance $d1$ to the center of the sub-box in the first run, and has a distance $d2$ to the center of the sub-box in the second run, then the weight on its computed value from the first runwill be $d2/(d1 + d2)$, and the weight on its computed value from the second run will be $d1/(d1 + d2)$. Figure 3.6 top right shows the weighted result. We can see from this figure that simple dividing strategy will introduce large errors in the borders. The weighted average method produces smaller errors overall and smoother-looking effect of the errors in the borders.

3.1.5 Further Compression-Run Length Encoding

	Size(in bytes)			
	Salinity	Temperature	Dissolved oxygen	Apparent oxygen uti.
Compr.xyz(RLE)	5983	3913	3994	4904
Compr.v	21394	17595	17046	19871
compr.(xyz+v)	27377	21508	21040	24775
Bzip2	36733	25687	24082	27829
Size Reduction(%)	34.17%	19.42%	14.45%	12.32%

	Size(in bytes)			
	Percentage oxygen satura.	Phosphate	Nitrate	Silicate
Compr.xyz(RLE)	5026	5026	6118	4129
Compr.v	21717	19417	25787	18947
compr.(xyz+v)	26743	24493	31905	23076
Bzip2	35649	27658	39844	31018
Size Reduction(%)	33.30%	12.92%	24.88%	34.41%

Table 3.1: File size comparison after compression between the Run length encoding method and simply storing quadruplets which contains three coordinates and values. "Compr.xyz(RLE)" is the size of compressed coordinates using RLE and bizp2. "Compr.v" is the size of compressed scalar value using RLE and bizp2. "Bzip2" is the size of compressed 3D dataset using only bizp2. Our RLE method further reduces file size ranging from 12% to 34% as indicated in the fourth row

The result from the above steps of 3D-ODETLAP method is a representative sample of data points in 3D space. In 3D-ODETLAP, the compression result is a set of 3D coordinates *(x,y,z)* and their associated scalar value v. The coordinates *(x,y,z)* are different from value v because they distribute evenly within the range in each dimension respectively in our test data. But values v are distributed more closely and require higher precision for storage.

The run-length encoding is a simple lossless compression technique and it stores the value and the count of sequence with the same value. Because the coordinates *(x,y,z)* correspond to positions in a 3D matrix, we only need to store a binary value in each position to indicate whether this point is selected or not. So we define *Run* as a consecutive sequence of 0's or 1's to represent the selection of one point at each location. Thus, given a binary matrix of $N1 \times N2 \times N3$, the method is as the following: for each *run* length L, test if

1. $L < 254$, then store it in one byte
2. $254 \leq L < 510$, then use FFFE as a marker and a second byte for L - 254
3. $510 \leq L < 766$, then use FFFF as a marker and a second byte for L - 510
4. $L \geq 766$, then use FFFFFFFF as a two byte marker and use next two bytes for L.

Generally, most runs are below 254 and this means for most runs we only need one byte to store it. Here we assume all runs are shorter than 65535, which works well in our test datasets. After RLE, we apply bizp2 on the compressed coordinates and scalar value v to produce the final compressed file. This simple prefix-coding further compresses the sparse sample points as shown in Table 3.1.

3.2 5D-ODETLAP

3.2.1 Definition

5D-ODETLAP, or Five Dimensional Over-Determined Laplacian Partial Differential Equation, is an extension of the Laplacian PDE $\frac{\delta^2 z}{\delta x^2} + \frac{\delta^2 z}{\delta y^2} = 0$ to an overde-

termined linear system [54,62]. It is the core part of our 5D-ODETLAP compression framework.

In this overdetermined linear system, every point, known or unknown, has an equation setting its value as the average of its 4, 5, 6, 7 or 8 neighbors in four dimensional space. The equation is:

$$\begin{aligned} u_{i,j,k,t} = (&u_{i-1,j,k,t} + u_{i+1,j,k,t} + u_{i,j-1,k,t} \\ &+u_{i,j+1,k,t} + u_{i,j,k-1,t} + u_{i,j,k+1,t} \\ &+u_{i,j,k,t-1} + u_{i,j,k,t+1})/8 \end{aligned} \tag{3.6}$$

for every point. Here $u_{i-1,j,k,t}$ is the immediate neighbor of $u_{i,j,k,t}$ in the first dimension and so forth.

Unfortunately, this simple linear system will have one exact solution, which is probably not the optimal one for different data distributions. On the other hand, the solution to the Laplace equation doesn't have a relative maximum or minimum in the interior of the solution domain, which is defined as the maximum principle [52].

In order to generate local maximum/minimum values, we first apply the Equation 3.6 to every non-border point and then add one equation for each known point in a set S:

$$u_{i,j,k,t} = h_{i,j,k,t} \tag{3.7}$$

where $h_{i,j,k,t}$ stands for the known value of points in S and $u_{i,j,k,t}$ is the "computed" value as in Equation 3.6. So now we have more equations than points in the data, and this means the linear system is over-determined. This 5D-ODETLAP is essentially different from the Laplacian PDE because the Laplacian PDE only assigns one average equation to every data point, while 5D-ODETLAP assigns one exact equation and one average equation to every known data point. So for each known data point, its value is equal to two different values, which makes 5D-ODETLAP an over-determined linear system and have completely different mathematical properties. For example, 5D-ODETLAP has the advantage of being able to infer a local extremum at an unknown point, which is impossible with a Laplacian. Another advantage of 5D-ODETLAP is that slopes are continuous across lines of known points,

which reduces artifacts in the generated surface.

Also, the least squares solution can be biased at certain data points by multiplying both sides of Equation 3.6 for those points a by parameter R. Since the solution to this system may not be consistent, we obtain an approximate solution instead of an exact solution (which is impossible) by keeping the error as small as possible. Equation 3.1 makes the data distribution in 4D more "smooth" and continuous, while Equation 3.7 keeps the value at each known point close to its known value. So for every known point, we can choose the relative importance of data continuity versus accuracy through R.

If R is small, the system will approximate a deterministic solution favoring data accuracy. If R is very large, the system will produce a surface with no divergence (in the vector calculus sense), ignoring the known values. In [33] and [32], $R = 0.01$ was found to be a good choice. Since a 5D dataset is a sequence of 4D volumes, we apply the above approximation on each 4D volume and then refine it in 5D to minimize the compression error.

As a result, a very small representative representative sample of the 5D dataset is sufficient to achieve a high compression ratio together with a reasonable error after further encoding.

3.2.2 Framework Outline

An outline of the overall 5D-ODETLAP framework is given in Figure 3.7. Beginning with a 5D dataset, we use a uniform random point selection to obtain a sample point set S. Each point has an equal probability of selection. Then an initial approximation to the 5D data is computed using Equations 3.6 and 3.7. This approximation is obtained by individually applying 4D reconstruction on each 4D slice of the 5D dataset and then merging them back together to form a complete 5D approximation. Then we require users to set a stopping condition based on an error metric. In practice, we use a user-defined average mean percentage error of all 4D datasets within the 5D dataset as the stopping condition, because each 4D dataset has a different data range and represents completely different practical meaning in geospatial fields. For example, if the reconstructed 5D dataset consists of three 4D

slices with mean percentage errors of 1%, 2% and 3% respectively, the average mean percentage error for this whole 5D dataset will be $(1 + 2 + 3)/3 = 2\%$. Intuitively, the mean percentage error of a whole 5D approximation will be the average of the mean error of all 4D slices.

If the stopping condition is not satisfied, we calculate the percentage errors of all the points in the previous 5D approximation. The percentage errors are based on the data range of each 4D slice with different environmental variables. So in this way, we do not bias any 4D slice. We then select k ($k \geq 1$) points with the largest percent error with a restriction called *Four Dimension Forbidden Zone* to optimize our selection. These selected points will be added into the existing sample set S, and this extended set is used again to compute a more refined 5D approximation by applying 4D reconstruction on each 4D dataset and putting them back together. Again, this 5D approximation will be evaluated according to the error metric to see if the stopping condition is satisfied. These two steps work alternatively as the algorithm proceeds. A better approximation is obtained as the size of point set S increases and the error shrinks. Note that the key point in 5D compression is to pick the points with largest errors. The 4D reconstruction is done individually for each 4D dataset, and this step is used to evaluate the compression from the previous iteration. The reason for this design pattern is to make our method suitable for compressing 5D data which contains different variables in 4D space.

```
Input: 5D − GeospatialData : V
Output: PointSet : S
S = InitialSelection(V);
foreach 4D volumes s in S do
    vReconstructed = 4DReconstruction(s);
    add vReconstructed to set VReconstructed;
end
while MeanError > Max_MeanError do
    S = S ∪ Refine(V, VReconstructed);
    foreach 4D volumes s in S do
        vReconstructed = 4DReconstruction(s);
        add vReconstructed in set VReconstructed;
    end
end
return S
```

Algorithm 2: Iterative sampling of 5D-ODETLAP framework pseudo code

The pseudo code in Algorithm 2 demonstrates the above iterative sampling process of 5D-ODETLAP. After reaching the stopping condition, further encoding of the set S is done by splitting this set S into a binary 5D matrix with point position information and a floating point value vector. We will discuss the further encoding later.

3.2.3 Four Dimension Forbidden Zone

Four Dimension Forbidden Zone is a natural extension of the *Forbidden Zone* method in 3D-ODETLAP. The naive way of selecting refined points would be to pick the first k (k≥1) points with largest percentage error. But this strategy needs further improvement in that those selected points are often clustered due to abrupt value change within a small 4D region. In our experiment, if one point with large error is selected, it's most likely that its immediate neighbors are erroneous as well and thus will be selected. So this introduces redundancy in some regions, which we need to minimize in order to achieve a high compression ratio.

Reminiscent of the *Zone of Inhibition* in ecology, the *Four Dimension Forbidden Zone* is the restriction we put upon the process of adding new refined points:

the candidate point's spatial local neighbors in 4D space will be checked to test if there is one or more existing refined points added in the same iteration. If yes, this candidate point is abandoned and the point with the next biggest error is tested until we add a predefined number of points. Figure 3.8 is a 2D illustration of a forbidden zone. The red point has been added in this iteration, and the circle around it is the forbidden zone. The green points are in the circle so they are not included. All the blue points are outside the zone and thus have the possibility to be included in this iteration if they are not in other added points' forbidden zone. An optimal forbidden zone size would be related to the data's spatial autocorrelation. However, this affects only the algorithm's speed, we simply pick a zone size experimentally.

3.2.4 Four Dimension Dividing into Boxes

We extend the *Dividing into Boxes* method in 3D-ODETLAP to 4D. The difference between them is the size of the smaller boxes. In 3D-ODETLAP, it's a 3D boxes, while it grows to be a 4D boxes divided from the original 4D dataset in 5D-ODETLAP. The Figure 3.9 illustrates this method in 2D.

3.2.5 Initial Point Selection

	Data1			
Method	Random1	Random2	Grid	Border
Points	10527	10227	10356	10344
Mean Err(%)	2.240	2.302	3.040	2.460
Max Err(%)	36.33	46.67	58.76	49.60
	Data2			
Method	Random1	Random2	Grid	Border
Points	14040	14058	13968	14088
Mean Err(%)	2.341	2.354	3.610	3.066
Max Err(%)	59.360	59.636	58.358	59.900

Table 3.2: Results of initial approximation from using different initial point selection methods on two datasets. Data1 has 99.82% filled points and Data2 has only 58.01% filled points. Random1 and Random2 are two runs on the same dataset using Matlab's uniformly random method.

The difference in initial point selection strategies may result in large differences in the quality of consequent approximations. Our above *Four Dimension*

Dividing into Boxes strategy involves doing reconstruction in smaller 4D sub-boxes and then merging them back together to form a complete 4D reconstruction. So it's inevitable that on the borders of these small 4D sub-boxes, we don't have enough information from the data to produce a good approximation even though we alleviate this problem by taking weighted average of two different recontracted values at one point [33]. To have a good initial approximation, we created a *Border Selection* initial point selection method, which gives priority to points on borders of smaller sub-boxes, and then use regular grid selection inside each sub-boxes to get the rest of points initially. Then we compared this method with the regular grid selection and the uniformly random selection method by keeping the number of initially selected point to be the same. This uniformly random selection method ensure that each point has an equal opportunity to be selected.

We can see from Table 3.2 that the uniformly random selection method actually has a surprisingly smaller mean and max percentage error on both datasets. Although two runs of random selection have slightly different results, this discrepancy is trivial as shown in the Table 3.2. The number of points selected by all methods are approximately the same: in data1, mean value is 10,364 and standard deviation is only 107.01; in data2, mean value is 14,039 and standard deviation is only 44.17. Uniformly random selection methods in general are better than regular grid and *Border Selection* methods. In fact, the two runs are a Monte Carlo simulation with $N = 2$. Note that since each 4D dataset in the 5D datset has different data range and represents different environmental variables, we take the average of all 4D datasets' mean percentage error as the mean percentage error for the whole 5D dataset. Similarly, we also take maximum of all 4D datasets' maximum percentage error as the maximum percentage error for the whole 5D dataset. This metric is used in the entire thesis.

3.2.6 Further Encoding

3.2.6.1 Algorithm Outline

After the iterative greedy sampling process, we have a representative sample of the original 5D data. As shown in the Figure 3.10, the further encoding starts with

splitting the sparse sample set of the original 5D dataset into a binary 5D dataset giving the positions of the selected points and a floating point vector containing the scalar value of the selected points in row-major order. For the binary 5D position matrix, we first apply our *Binary RLE* (Run Length Encoding) to change it to a vector of integers. Then we use RLE and a Prediction by Partial Matching (RLE_PPM) coder for entropy encoding of the integer vector to get a final compressed file. Since the position information of each selected point must be reconstructed exactly during the decompression, no information loss is allowed.

However, for the vector of scalars, information loss is tolerable. We first apply Lloyd [34] quantization to produce an optimal quantization of the floating point value with a user input, which is the number of bits used for each float number. This ranges from 6 to 12 bits. After the quantization, we have a resulting integer vector and a small floating vector called codebook of size 2^k, where k is the number of bits in the input. Each member of this floating vector indicates the assigned value for the inputs that fall into the range of that partition in the dequantization process. Then we apply delta encoding to further reduce the entropy of this integer vector. At this step, the delta encoding is conditional because it adds one sign bit, which change the previously unsigned integer vector into signed integer vector. This process will affect the performance of the next step, so it is conditional. Finally we apply a PPM coder on both the resulting integer vector file and the codebook file.

We also used LEDA, a C++ class library for efficient data types and algorithms [40], to build our own entropy coder for the further encoding part of the 5D-ODETLAP compression framework. LEDA contains a wide range of simple coders, which could be easily integrated together for compressing data with special structure. We built ten different coders using LEDA to find the one that best suits our needs. They are the following:

- A0: Adaptive Arithmetic Coding
- BMRA: Burrows-Wheeler Transform + Move to Front + RLE for Runs of Zeros + Adaptive Arithmetic Coding
- BRA: Burrows-Wheeler Transform + RLE for Runs of Zeros + Adaptive

Arithmetic Coding

- Huff: Adaptive Huffman Coding
- MRP: Move to Front + RLE for Runs of Zeros + Prediction by Partial Matching
- RMP: RLE for Runs of Zeros + Move to Front + Prediction by Partial Matching
- RDP: RLE for Runs of Zeros + Dictionary-based Coder + Prediction by Partial Matching
- PPM: Prediction by Partial Matching
- Dict_PPM: Dictionary-based Coder + Prediction by Partial Matching
- RLE_PPM: RLE for Runs of Zeros + Prediction by Partial Matching

3.2.6.2 Binary Position Compression

For the sparse binary 5D matrix, first we use our modified RLE method, *Binary RLE*, to change it into a integer vector. The traditional RLE stores a data value followed by the number of the consecutive occurrences. Instead, our *Binary RLE* is defined as:

$$P = \{r_1 r_2 \dots r_i \dots r_k\} \tag{3.8}$$

where P is the resulting integer vector and r_i represents the number of 0's before next 1 in a row-major order of the binary 5D matrix. k is the total number of selected points, namely, the number of 1's. The main reason for the *Binary RLE* is that our input is the 5D binary matrix, so the number is either 0 or 1. There is no need to store the symbol number.

Figure 3.11 shows the result of using 17 different coders to compress the 5D binary position file. The 10 coders from LEDA as shown in List 3.2.6.1 compress the integer vector from *Binary RLE*. We also include the result of the Bzip2 [51] and JPEG 2000 coder for the integer vector. The International Telegraph and Telephone Consultative Committee (CCITT) Fax3, CCITTFax4, Lempel-Ziv-Welch (LZW)

[45], PackBits [6] and Deflate methods are the five coders in Tagged Image File Format (TIFF). Since TIFF has a well-known excellent performance in compressing binary image files, we store the binary 5D matrix as a black\white image using the above 5 different coders in TIFF for comparison purpose. The test data is $90 \times 180 \times 24 \times 12 \times 4$ from WOA 2009 data. The number of selected points is 91119. Clearly, RLE_PPM coder gives us the highest compression ratio in this case and remains the best in other test cases, so we use this coder for our 5D-ODETLAP framework.

3.2.6.3 Floating Value Compression

For the vector of scalar value, we use Lloyd relaxation to produce a optimal quantization for the floating point value. The best-known and earliest quantization algorithm, generalized Lloyd algorithm (GLA) [34] is based on minimizing the Mean Square Quantization error (MSE_q), which can be expressed as:

$$MSE_q = \sum_{i=1}^{N} \int_{d_i}^{d_{i+1}} (x - C(x))^2 f_x(x) dx \tag{3.9}$$

where x and $C(x)$ are input floating values and the quantized output, respectively. N is the total number of reconstruction levels, and $f_x(x)$ is the probability density function (pdf) of the input vector. The quantization give us two files: one contains the integer index file and another floating point codebook file.

After the quantization step, we apply a conditional delta encoding to the quantized integer index, because the delta encoding adds one sign bit to the integer vector, and this can affect the performance of our next step in the compression process as shown in Table 3.3. That is because the PPM coder we use as the next step in the compression step takes the input as a byte stream. For example, if we apply delta encoding and the input number of bits is 7. The delta encoding adds one sign bit for each number, which make it to be 8 bits per quantized integer. Under this situation, the delta encoding works better than the case when the input number of bits is 8. In fact, if the number of bits used to store the quantized integer is a multiple of 8 (one byte), or a multiple of 4 in generalization, the following PPM

Integer Range	Delta (bytes)	Non-Delta (bytes)	Advantage (%)
0-8 (3 bits)	10790	11132	−3.170
0-16 (4 bits)	23497	18750	20.203
0-32 (5 bits)	32882	32388	1.502
0-64 (6 bits)	48281	44236	8.378
0-128 (7 bits)	50916	58596	−15.084
0-256 (8 bits)	72132	59712	17.218
0-512 (9 bits)	83368	82611	0.916
0-1024 (10 bits)	97796	93932	3.951

Table 3.3: Influence of delta encoding on the size of the final compressed floating point value vector. Column "Advantage" indicates the advantage of delta encoding over non-delta encoding on the final result. Test data is the same as in Figure 3.11.

coder works the best. So when the input number of bits is 3 or 7, we use delta encoding.

Figure 3.12 shows the result of using different coders to compress the quantized floating point vector without delta encoding. The input number of bits used for each value is 8. The test data is the same as in Figure 3.11. We also include the result of the Bzip2 coder for comparison. This figure demonstrates that PPM coder performs the best in our 5D-ODETLAP compression framework.

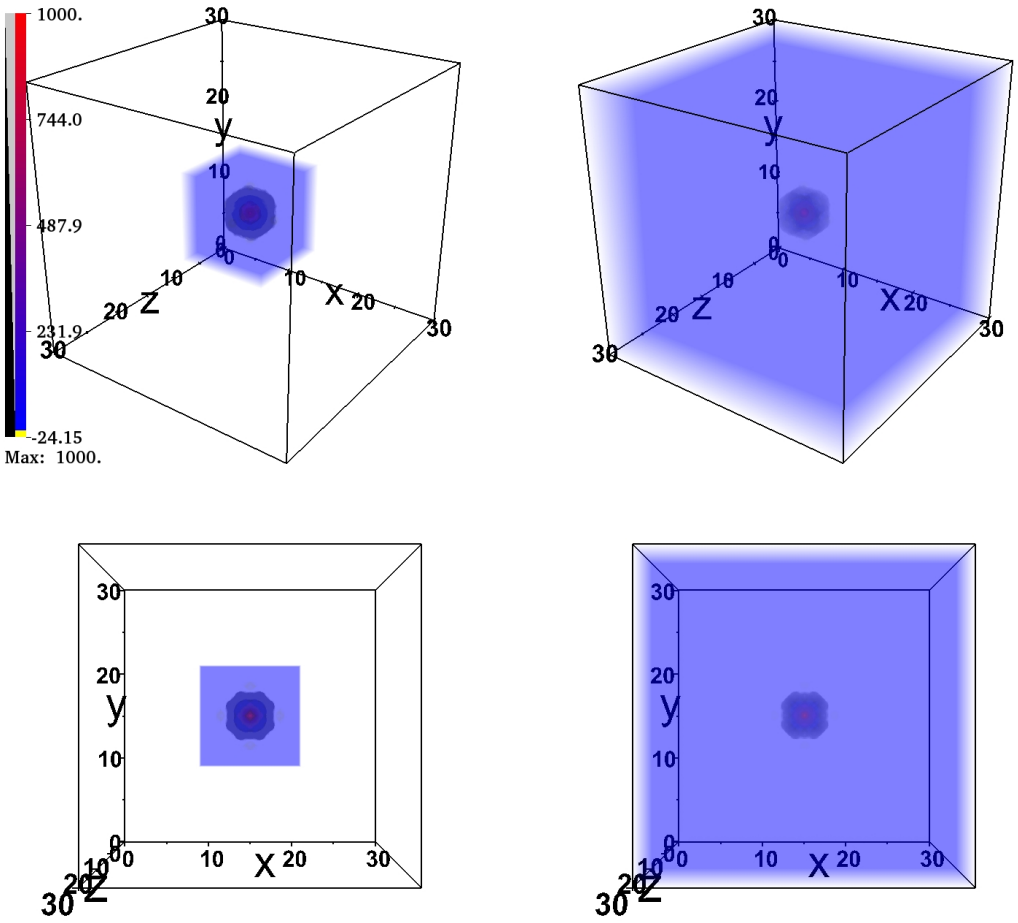

Figure 3.5: The limited radius of impact during reconstruction from altering a single known point in the data. This plot shows the per-point difference between the 3D-ODETLAP solution for a $30 \times 30 \times 30$ 3D matrix with value equal to 1 and the ODETLAP solution for the same data with the central point's value modified to 1000. The difference in the solutions is restricted to a $15 \times 15 \times 15$ area around the altered point, with 88% of the affected points concentrated in the center $10 \times 10 \times 10$ area.

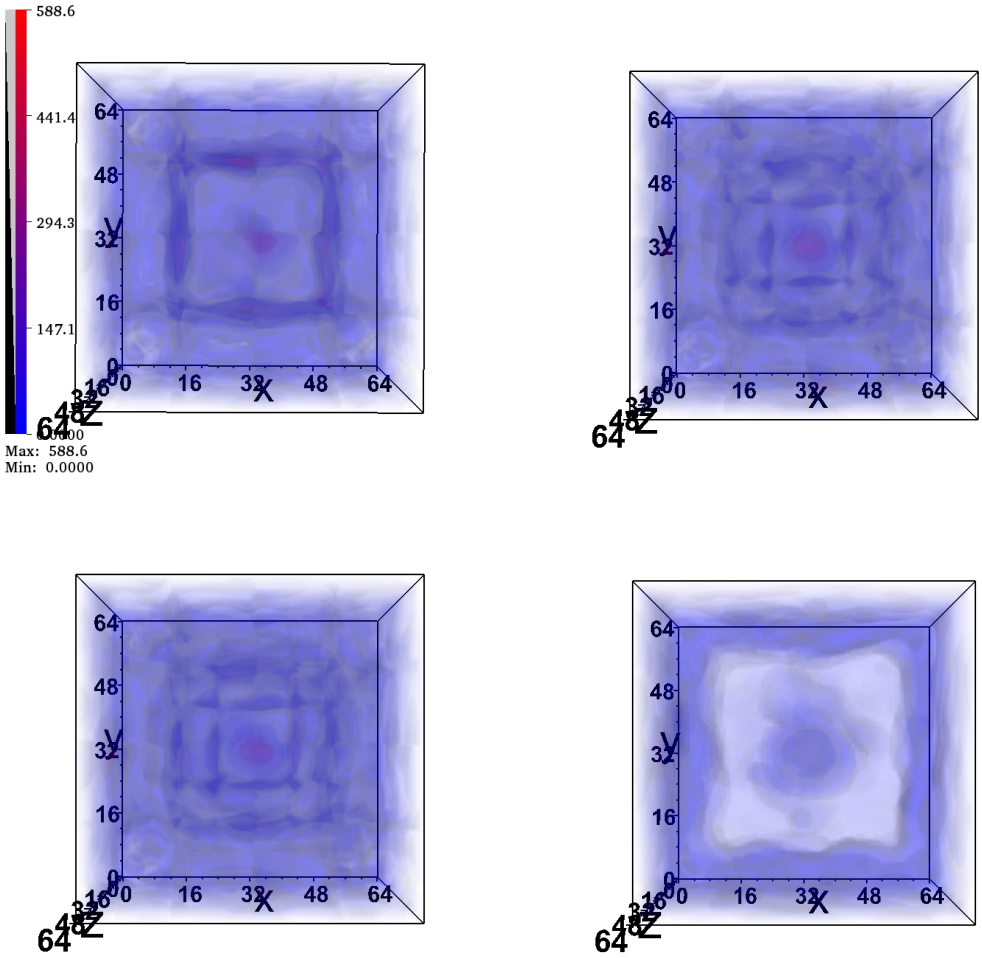

Figure 3.6: Visual effect of different dividing strategies. Top left: Error of Simple Dividing Strategy; Bottom left: Error of Averaging Overlapping Area Strategy; Top right: Error of Weighted Averaging Strategy; Bottom right: Error of non-dividing Strategy

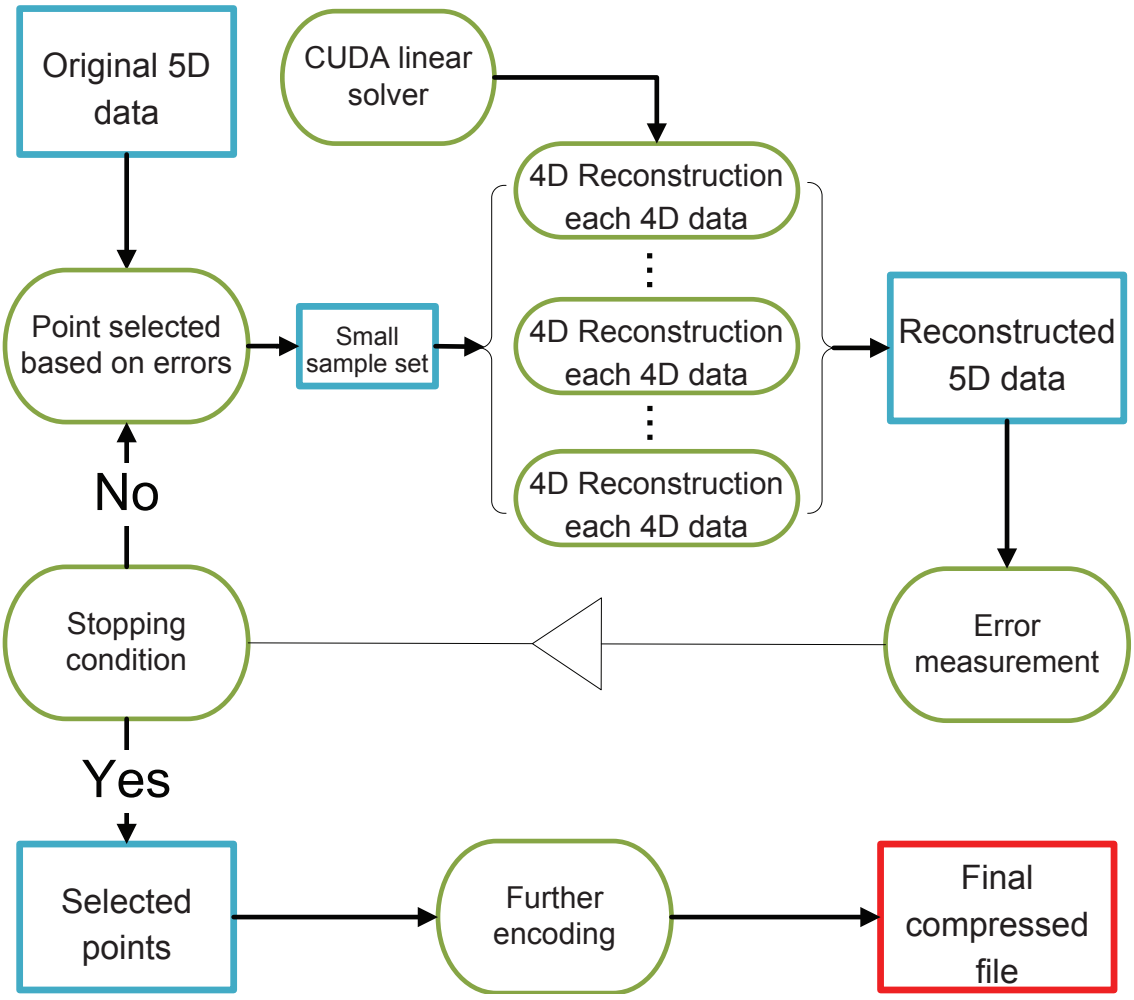

Figure 3.7: 5D-ODETLAP framework outline.

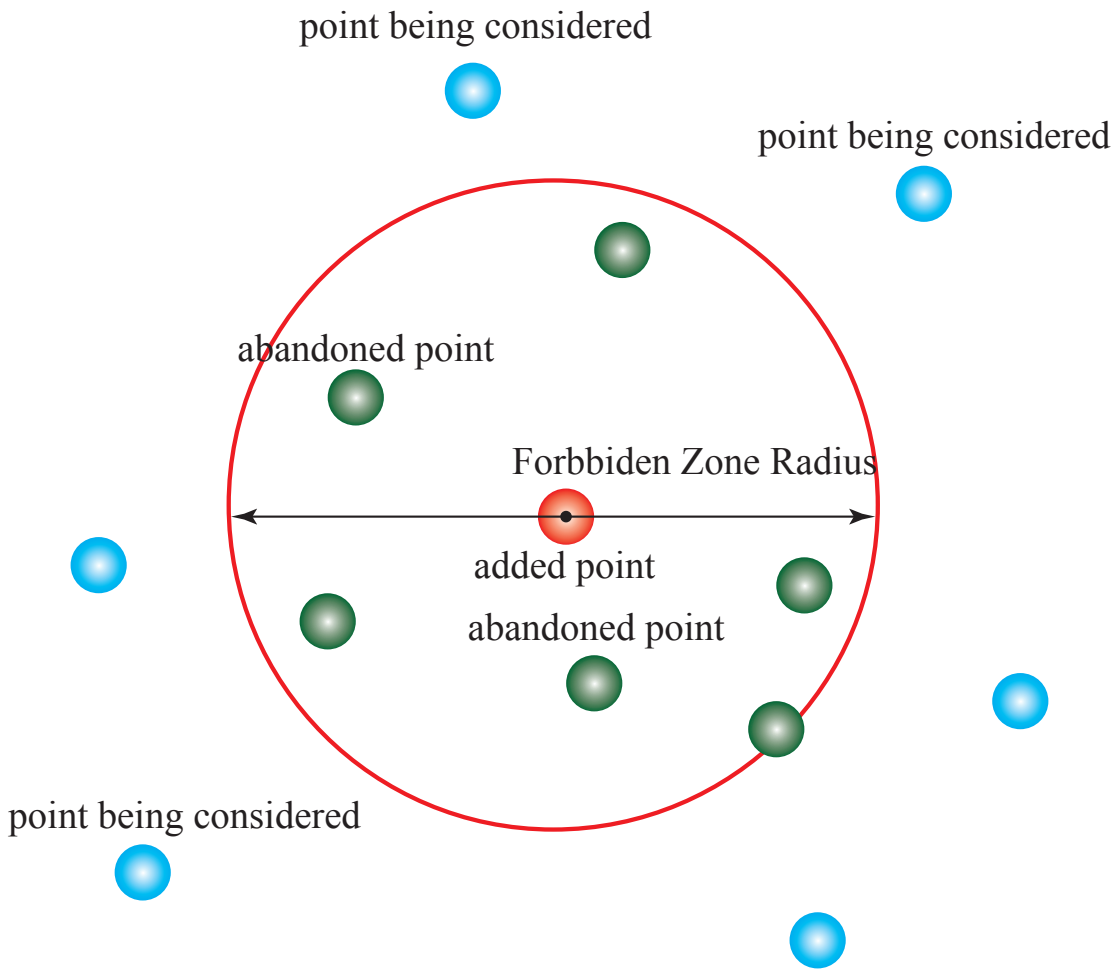

Figure 3.8: 4D Forbidden Zone check. The red sphere represents the forbidden zone of the red point in the center. The green point are abandoned. The blue ones are being considered.

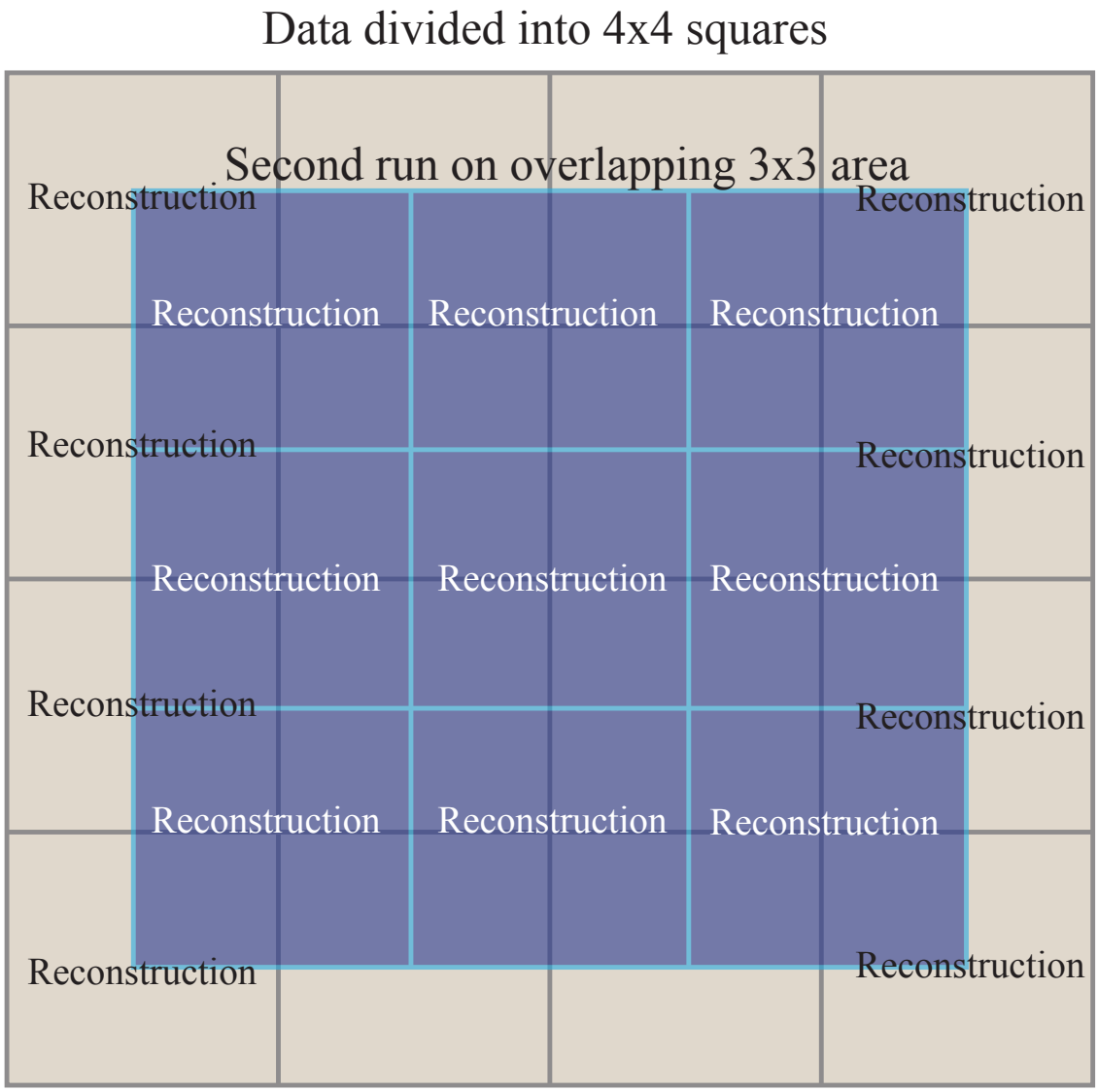

Figure 3.9: 2D illustration of *Four Dimension Dividing into Boxes* method. Reconstruction is done individually for each square in first run as marked by grey squares. The 50% overlapping blue squares also have computed value for each point from the second run, and we take a weighted average of both values in the overlapping area (blue) to produce a good approximation.

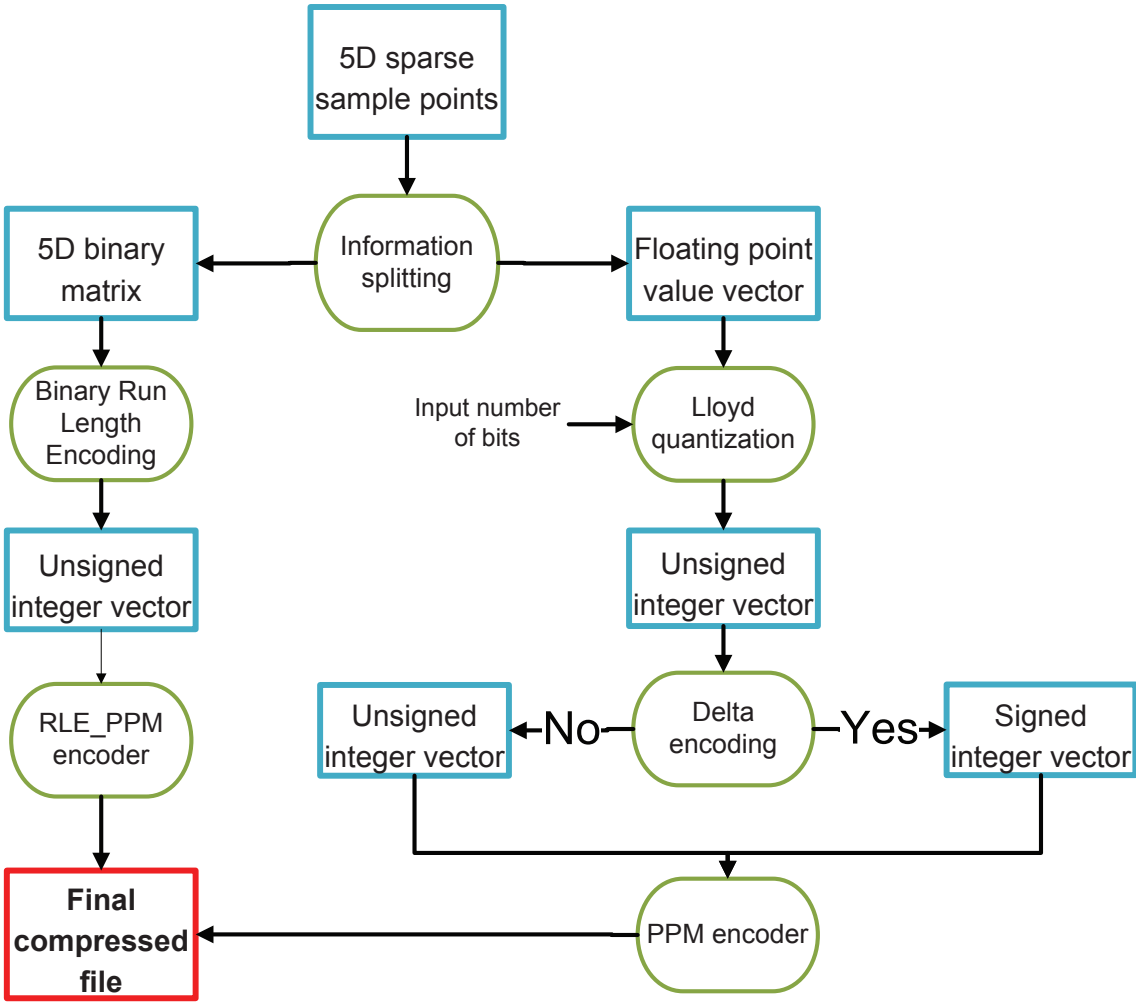

Figure 3.10: Outline of the further encoding part of the 5D-ODETLAP compression framework in Figure 3.7.

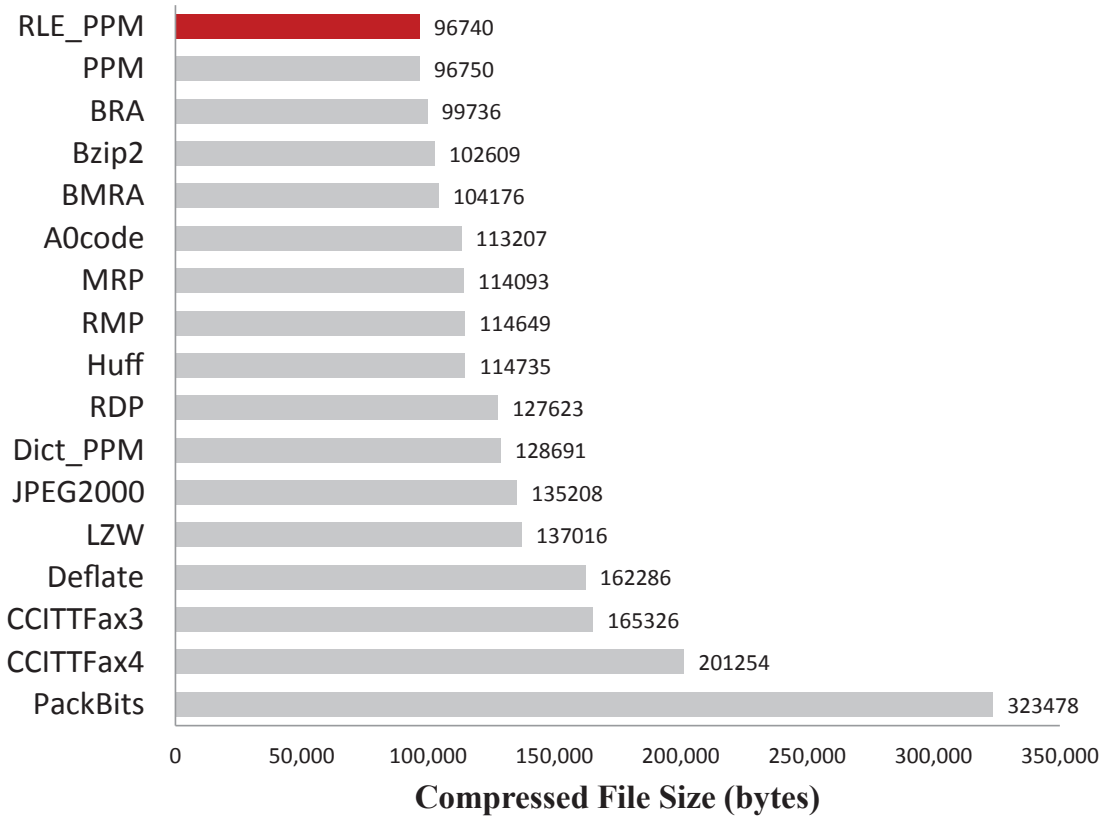

Figure 3.11: Binary position file compressed file size by 17 different coders. The uncompressed 5D binary position file is 2332800 bytes.

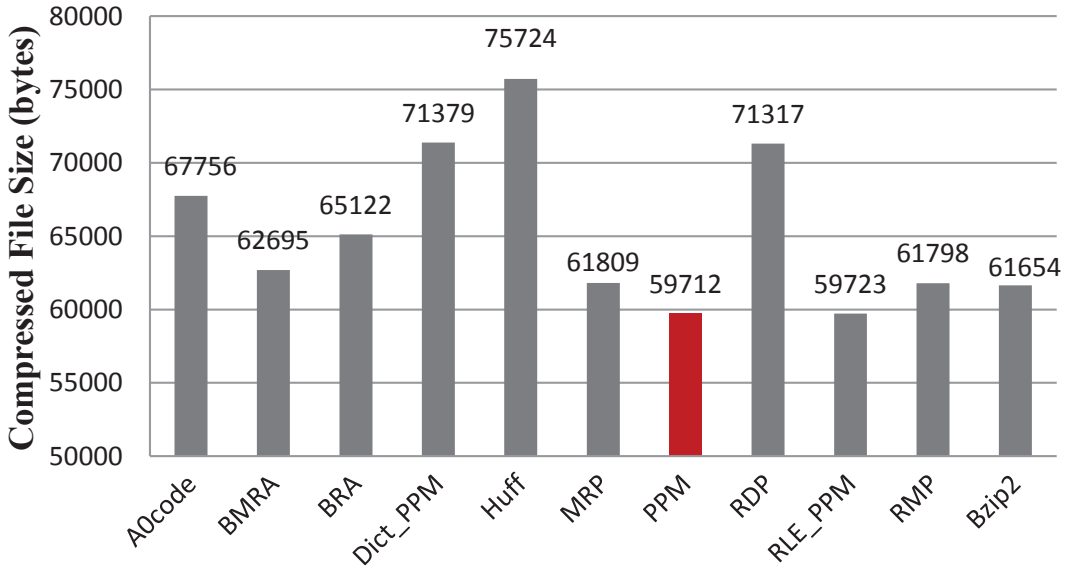

Figure 3.12: Floating point value vector final compressed file size by 11 different coders.

4. CUDA SOLVER FOR HD-ODETLAP

4.1 Solver Introduction

Even after the *Normal Equations* transform, solving the $A^T Ax = A^T b$ sparse linear system is still too slow and big. For example, solving a 104976×104976 linear system can take 58 GB main memory and 1.8 CPU hours on a workstation with four 2.4GHz processors and 60 GB of main memory running Ubuntu 10.04.2 and 64-bit Matlab R2009a. Nevertheless, solving this system is an indispensable part of the 5D-ODETLAP method and occupies the main computational cost. Our previous CPU-based linear solver proved to be the performance bottleneck for 5D-ODETLAP.

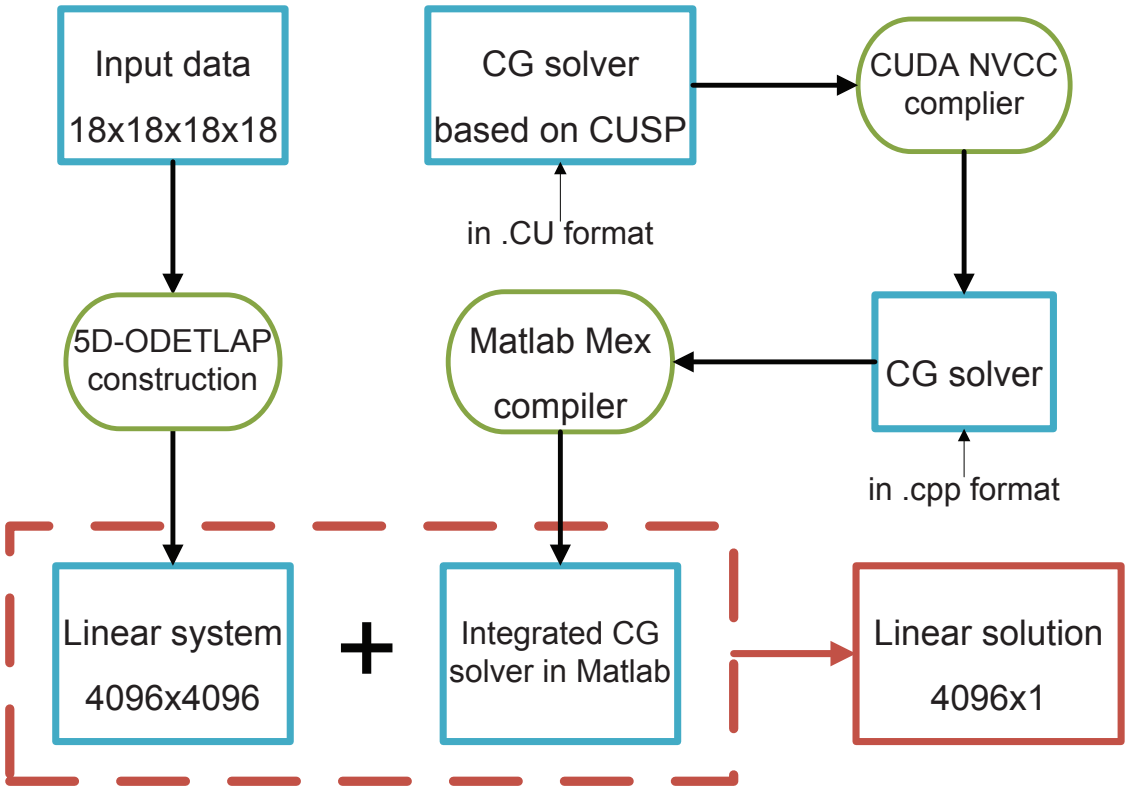

Figure 4.1: Outline of CUSP integration into Matlab.

CUDA by Nvidia is a developer-friendly interface for General-Purpose computation on Graphics Processing Units (GPGPU) [37]. It has been widely used to

perform general purpose computation of non-graphics data on GPUs to take advantage of the enormous computing power of stream processors. CUSP [8] is a open source library for sparse linear algebra computations on CUDA. It provides a flexible, high-level interface for manipulating sparse matrices and solving sparse linear systems. It is an open source project hosted on Google code by the author of [7]. The implementation of CUSP is mainly based on a set of CUDA-based fast Sparse matrix-vector multiplication (SpMV) routines developed in [7]. These routines are, in turn, based on a template library (Thrust), which resembles the C++ standard template library, but uses GPUs.

This library provides multiple sparse matrix formats as follows:

- COO - Coordinate
- Compressed Sparse Row
- DIA - Diagonal
- ELL - ELLPACK/ITPACK
- HYB - Hybrid

Users can incorporate CUSP into their own code using any of these formats and fill them with necessary data. Efficient conversions between these format, matrix file I/O and simple matrix operations (including matrix transpose and matrix-vector multiplication) are also available as a simple function call. In addition, it provides a Jacobi preconditioner for possible speed-up of certain linear systems. Most importantly, CUSP contains two iterative linear solvers, Conjugate Gradient solver (CG) and Biconjugate Gradient Stabilized solver (BICGS). With proper construction of a linear system, a simple function call can solve the linear system with GPUs without any knowledge of CUDA GPU programming. Appendix A shows a simple CUSP problem using CG solver.

4.2 Solver Selection and Matlab Integration

Equation 3.4 indicates a square, symmetric and positive-definite linear system. Figure 4.2 illustrates the nonzero entries of the A^TA matrix. The current Matlab CF

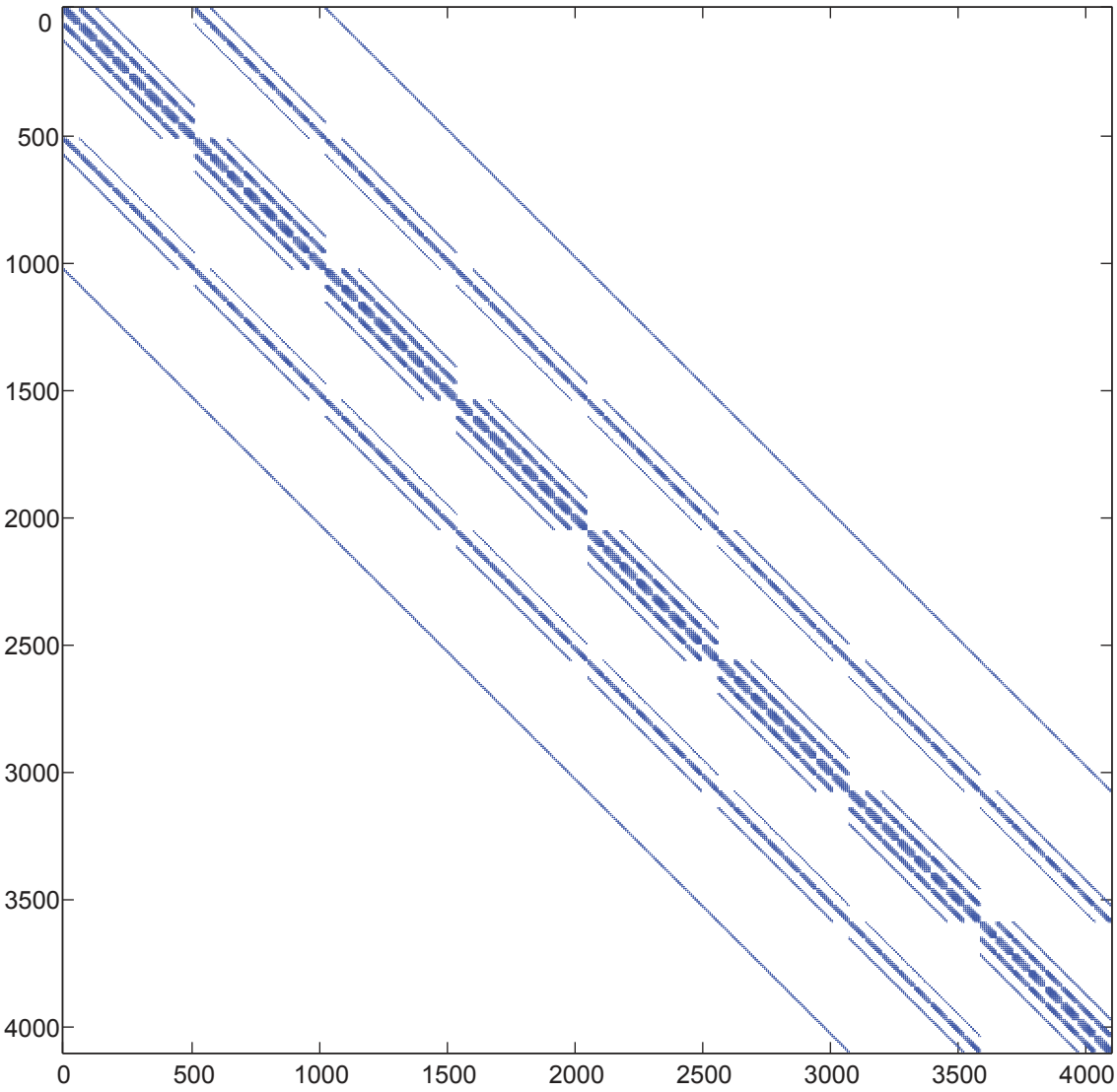

Figure 4.2: A 4096×4096 square, symmetric and positive-definite linear system in 5D-ODETLAP. Number of non-zeros is 132608.

solver solves this linear system directly to maximum machine precision. That is both compute-intensive and unnecessary because such a high precision is not required. 5D-ODETLAP is a lossy compression method at a certain error level. Given the very developer-friendly interface of CUDA, as illustrated in Appendix A, the CUSP CG iterative solver is a natural replacement.

The test data in Table 4.1 is created using the function $F = i^2 + j^2 + k^2 + q^2$, where *i, j, k* and *q* are the coordinates of one point in the 4D matrix. In this data, 0.3% of uniformly randomly selected points are selected as the known points. The

System Size	CPU-based Cholesky factor solver (s)	CUSP Solver w/o overhead (s)	CUSP w. overhead (s)	**Ratio of CF to CUSP w/o overhead**
4096×4096	2.88	0.16	2.72	18.0
6561×6561	8.30	0.27	4.54	30.7
10000×10000	20.11	0.37	7.03	54.4
14641×14641	48.57	0.56	11.05	86.7
20736×20736	106.99	0.64	14.84	167.2
28561×28561	215.79	0.85	20.61	253.9
38416×38416	471.91	1.28	28.22	368.7
50625×50625	844.96	1.77	38.31	477.4
65536×65536	1844.54	2.35	48.45	784.9
83521×83521	3346.02	3.00	62.26	1115.3
104976×104976	6692.70	4.35	79.61	1538.6
130321×130321	9678.98	5.32	99.93	1819.4
160000×160000	21368.83	6.82	120.90	3133.3
194481×194481	31761.16	8.03	147.20	3955.3
234256×234256	49237.65	8.96	179.01	5495.3

Table 4.1: Efficiency improvement of overhead-free CUSP CG solver over Matlab CF direct solver using Cholesky Factorization. Elapsed time measurement is in seconds. Size of the data represents the size of the linear system from the 5D-ODETLAP method. The linear systems solved are constructed from the datasets in the table in 4D reconstruction of the 5D-ODETLAP method.

original data size in Table 4.1 ranges from 8^4 to 22^4, so the resulting linear system in 5D-ODETLAP ranges from 4096×4096 to 234256×234256. The CUSP solver is set to produce a solution with residual less or equal than 1e-6. Since the CUSP library is implemented in C++ and CUDA, we use file I/O to transfer the matrix data between 5D-ODETLAP and the CUSP library on a workstation with four 2.4GHz processors and 60 GB of main memory running Ubuntu 10.04.2 and 64-bit Matlab R2009a. For a fast implementation, we write the matrix in binary format through the *shared memory* in the Ubuntu system. The column "Cholesky Factor. Solver" records the running time of solving the different linear systems constructed from the test matrices. Column "CUSP Solver" only keeps record of the solving time on the CUSP library, while column "CUSP with Overhead" stores the time spent on

solving the linear system and transferring data between the Matlab program and the CUSP library, and the time of transferring data from main memory to GPU memory in the CUSP library.

From this table we can conclude that:

- the Matlab's CF direct solver is much slower than the CG solver in CUSP in general.
- the actual time in solving the matrix accounts for only 5.08% of the overall solving time on average, while the file I/O takes up a large fraction of it.

So this means the simple *shared memory* approach we used for transferring data between Matlab and the CUSP solver still cause too much overhead, which makes it necessary to integrate the CUSP library into our Matlab implementation in order to minimize this overhead.

System Size	Cholesky Factor. Solver (s)	Matlab CG (s)	CUSP CG (s)	**CUSP CG Jacobi (s)**	CUSP BICGS Jacobi (s)
4096^2	2.46	0.63	0.23	0.20	0.21
6561^2	6.49	1.11	0.56	0.30	0.36
10000^2	17.43	2.01	0.52	0.48	0.55
14641^2	43.02	3.02	0.75	0.70	0.77
20736^2	93.20	5.55	0.94	0.92	1.08
28561^2	203.25	5.40	1.29	1.28	1.32
38416^2	454.41	13.35	1.91	1.74	1.86
50625^2	816.90	15.58	2.28	2.20	2.34
65536^2	1791.78	29.30	2.89	2.54	3.03
83521^2	3332.05	30.67	3.57	3.57	3.41
104976^2	6488.24	39.67	4.86	4.84	5.14

Table 4.2: Elapsed time (in seconds) comparison between Matlab CF direct solver, Matlab CG solver, CUSP CG solver, CUSP CG solver with Jacobi preconditioner, CUSP BICGS solver with preconditioner and CUSP CG solver.

To reduce that overhead, we utilize the Matlab Executable (MEX) tool, which allows users to interface C, C++ or Fortran subroutines to MATLAB. MEX-files

are a way to call custom C, C++ or FORTRAN routines directly from MATLAB as if they were MATLAB built-in functions. In this way, we can largely reduce the overhead on transferring data between the Matlab program and the CUSP library. But since the CUSP library is implemented both in C++ and CUDA, this makes the MEX file more complex. Fortunately, the CUDA compiler allows users to compile CUDA code into C++ code as an intermediate step using the `nvcc --cuda` command. So. first we write the CUSP code in a MEX style as in Appendix B. Then this mixed source code is compiled into C++ code, which is the compiled into a MEX file and finally called directly from the Matlab program. The overall structure is outlined in Figure 4.1.

Appendix B provides the simplest program to call CUSP solver from Matlab to compute on GPU. Listing 4.1 shows the way of compiling this program in Matlab in Linux as shown in Figure 4.1.

Listing 4.1: Compiling CUSP code into MEX file

```
1  system(sprintf('nvcc -I"\%s/extern/include" --cuda "↩
       gpu_solver.cu" --output-file "gpu_solver.cpp"', ↩
       matlabroot));
2  mex -I/usr/local/cuda/include -L/usr/local/cuda/lib64 -↩
       lcudart gpu_solver.cpp
```

In Table 4.2, a comparison of solvers in Matlab and CUSP demonstrates that for the linear system from the 5D-ODETL-AP, **CUSP CG solver with Jacobi preconditioner** runs the fastest. The linear sysytems in this table are constructed from original 4D data of size ranging from 8^4 to 18^4 for the purpose of lossy compression, so the resulting linear systems' size ranges from 4096×4096 to 104976×104976. This solver runs more than seven times faster than its CPU counterpart with the same residual size of their solutions in the test linear system of size 104976×104976. And it's more than 1340 times faster than the CF direct solve in Matlab. Figure 4.3 illustrates this improvement in two different test datasets.

Not only is the CG solver with Jacobi preconditioner from CUSP better in terms of running time, it uses only 13.2 GB main memory and less than 512 MB

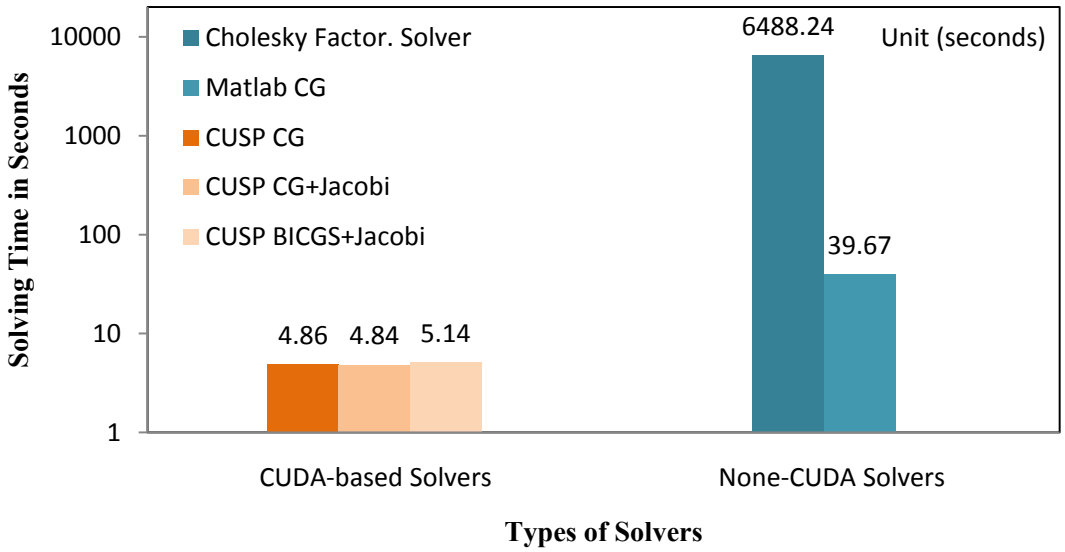

Figure 4.3: Performance comparison between CUDA solvers and Non-CUDA solvers in the 104976×104976 **test dataset from Table 4.2.**

device memory on GPU in our work station stated above.

4.3 Residual Size Test

Another problem in using this CUSP solver is to set the residual size, which is the error in the approximate solution to the linear system. To determine a suitable residual for our problem, we use a real world dataset, WOA 2005, to test how the mean and max percentage error change as the residual decreases. Table 4.3 shows that as the residual decreases, the mean and max percentage error stabilize under necessary precision. With a residual of 1.0×10^{-6} or 1.0×10^{-7}, the solution is sufficiently precise for our linear system. This test is done on the temperature data from World Ocean Atlas (WOA) 2005. The size of the data is $90 \times 120 \times 24 \times 12$, and it requires much more than 60GB of memory for a direct solution with CF in Matlab. So we use the CG solvers in Matlab so as to have a good comparison.

In Figure 4.4, we set a residual of 1.0×10^{-6} for the convergence of CUDA CG solver in the 5D-ODETLAP compression method. Then we also solve the same

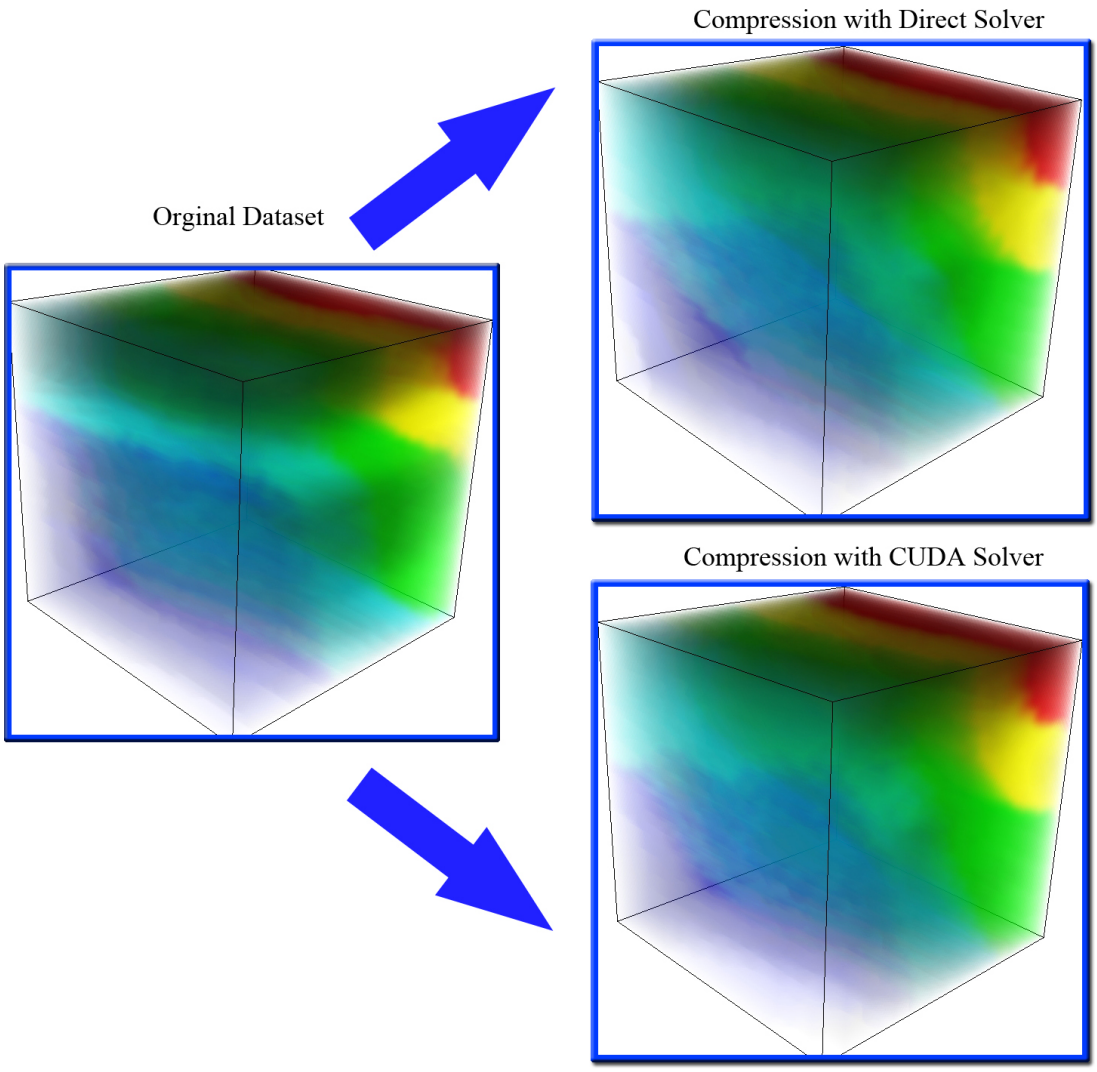

Figure 4.4: 3D illustration of compression results from Cholesky Factorization solver (Top Right) in Matlab and CUDA-based CG solver with Jacobi preconditioner(Bottom Right). The dataset is $16 \times 16 \times 16$ derived from the result of compressing a $16 \times 16 \times 16 \times 12 \times 4$ 5D dataset in WOA09. It represents one part of temperature data in Pacific Ocean in January, 2009.

Residual	CUSP Mean Err (%)	Matlab Mean Err(%)	CUSP Max Err (%)	Matlab Max Err (%)
10^{-1}	25.31	25.32	93.50	93.50
10^{-2}	25.31	25.32	93.50	93.50
10^{-3}	24.78	24.98	102.87	94.58
10^{-4}	2.80	3.10	99.27	99.03
10^{-5}	1.40	1.56	29.73	62.23
10^{-6}	1.36	1.38	23.49	32.06
10^{-7}	1.36	1.36	23.67	24.70
10^{-8}	1.36	1.36	23.65	24.48

Table 4.3: Mean and Maximum percentage error comparison between CUSP CG solver and Matlab CG solver. One iteration of 5D-ODETLAP is run for measurement of errors.

linear system using the CF direct solver from Matlab, which produces an exact solution. The compression ratio of both methods are intentionally set to be the same. However, the compression results of these two methods appear visually identical. Meanwhile, with a 2.6% mean percentage error and a compression ratio of 150:1, the reconstructed 3D images retain the overall structure as well as important details in the original image. This means using the CUDA CG solver will not affect the compression result either numerically and visually.

5. RESULTS AND ANALYSIS

5.1 3D-ODETLAP

5.1.1 Results and Analysis

Variable	Unit	Data Range	Size(MB)
Salinity	PPS	[5.00, 40.90]	8.16
Temperature	$^{\circ}C$	[-2.08, 29.78]	8.16
Dissolved oxygen	ml l^{-1}	[0, 9.55]	8.16
Apparent oxygen utilization	ml l^{-1}	[-1.43, 7.87]	8.16
Percent oxygen saturation	%	[0.31, 117.90]	8.16
Phosphate	μM	[0, 4.93]	8.16
Nitrate	μM	[0, 54.45]	8.16
Silicate	μM	[0, 256.24]	8.16

Table 5.1: Annual objectively analyzed climatology datasets on 33 standard depth levels on all 8 variables before compression. Each point uses 4 byte to store its value in single precision and thus total size of each dataset is 8.16 Mb.

Our test of 3D-ODETLAP is based on a real world marine 5D dataset–World Ocean Atlas 2005 (WOA05), which is provided by NODC (National Oceanographic Data Center). WOA05 is a set of objectively analyzed (1 degree grid) climatological fields of in situ *temperature, salinity, dissolved oxygen (Dis. Oxy.), Apparent Oxygen Utilization (A.O.U.), percent oxygen saturation (P.O.U.), phosphate, silicate,* and *nitrate* at standard depth levels for annual, seasonal, and monthly compositing periods for the World Ocean. It serves as an example of high dimensional geospatial datasets. Table5.1 shows the datasets in detail. In the following experiments, all of our test data are derived from these datasets.

In our implementation, several parameters can affect the effectiveness of our method, including initial point selection, points added at each iteration, smooth variable R and size of the forbidden zone. In order to achieve a high compression ratio while keeping the errors beneath a tolerant level, we approach as close as possible the optimal parameters for the all eight datasets by finding the best set of parameters for one dataset. This set of parameters works reasonably well with

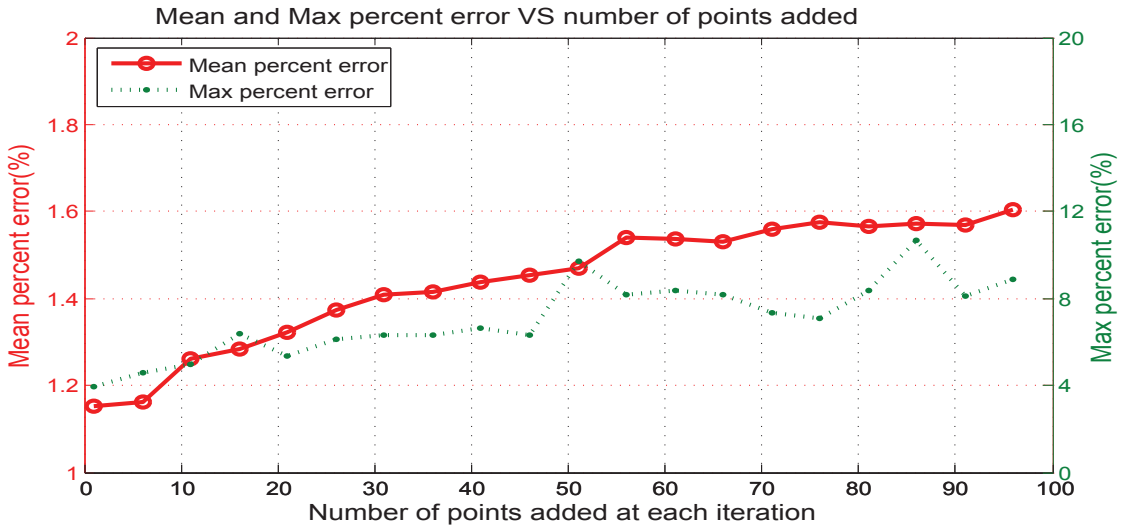

Figure 5.1: Mean(left) and Max(right) percentage error as the number of points added at each iteration increases from 1 to 96. The dataset is $64 \times 64 \times 32$ 3D matrix derived from Percent Oxygen Saturation in WOA 2005. The number of selected points is 2100, smooth variable R is set to 0.01, sub-box is $32 \times 32 \times 32$, initial point selection picks every 5 along the regular grid and the size of the forbidden zone is 3.

other datasets and if given enough computing resources, even better parameters can be found. Figure5.1 shows that as we increase the number of points added at each iteration, both mean error and max error are getting larger. But this doesn't mean we should keep this number to minimum, because there is still a trade-off between the computing time and the compression ratio with a predefined error.

Box Size	Mean Error(%)	Max Error(%)	Time (seconds)
$8 \times 8 \times 8$	1.8554	8.5103	150.40
$16 \times 16 \times 16$	1.3219	5.7181	293.90
$32 \times 32 \times 32$	1.1179	5.5175	1250.00

Table 5.2: Results of running 3D-ODETLAP with different sub-box sizes on part of Percent Oxygen Saturation dataset from surface to 4000 metres, which is a $128 \times 128 \times 32$ 3D matrix. Other parameters remain the same for all three tests.

Besides the above parameters, the size of each sub-box also has an impact on the compression performance. As table 5.2 shows, the larger the size of the sub-box, the better performance we will gain. However, the complexity of our algorithm

is extremely high as shown in figure 3.3, so we chose the largest possible sub-box while keeping running time at tolerant level. For our test on WOA 2005 data, a $30 \times 30 \times 30$ sub-box runs in about 2000 seconds at each iteration on four 2.4GHZ processors workstation with 60 GB of main memory running Ubuntu 9.04 and 64-bit Matlab R2009a. This is acceptable and we use a $30 \times 30 \times 30$ sub-box in all our tests, considering that there are only 33 levels of depth in WOA 2005.

Based on our implementation, we applied 3D-ODETLAP on the data in 5.1. This gave us eight distinct datasets in the size of $180 \times 360 \times 30$, which is reasonably large for testing purposes.

Variable	Mean Error(%)	Max Error(%)	Compress. File Size	Comp. Ratio
Salinity	0.1196	1.0870	60,037	130:1
Temperature	0.7283	4.1620	69,375	112:1
Dissolved Oxygen	1.3055	8.7205	65,501	119:1
Apparent Oxygen Utilization	1.4942	10.3178	59,019	132:1
Percent Oxygen Saturation	1.4930	8.3470	73,832	105:1
Phosphate.	1.0581	7.4413	62,693	124:1
Nitrate	1.3299	10.8315	71,060	109:1
Silicate	0.9969	9.8598	46,998	165:1

Table 5.3: Results of running 3D-ODETLATP on the 8 datasets derived from WOA 2005. Smooth variable R is set to 0.01, the size of a sub-box is $30 \times 30 \times 30$, initial point selection picks every 5 points along the regular grid, and refinement of point selection adds 20 points at each iteration and the size of the forbidden zone is 2. The size of each dataset is 7.42Mb with each point uses 4 bytes. For example, compressed Salinity dataset file size by 3D-ODETLAP is 60,037 bytes and thus has a compression ratio of 130:1.

In table 5.3, we show that with a tolerant absolute percentage mean and max error, 3D-ODETLAP can achieve a great compression ratio for all eight test datasets. Furthermore, the Silicate dataset from Table 5.3 and Table 5.6 has about the same mean percentage error: 0.9969% and 0.9996% respectively. But the compression ratio on the larger dataset is 165:1 while the one on the smaller dataset is only 81:1. This implies that 3D-ODETLAP may perform even better on larger datasets.

5.1.2 Comparison with 3D-SPIHT

We have reported on experiments using all eight large oceanographic datasets extracted from WOA 2005. The size of each dataset is $128 \times 128 \times 32$ with each point caring a single-precision scalar value, stored in 4 bytes each and resulting in a total size of 2 Mbytes. The same dataset is used for compression in our experiments with the 3D-SPIHT method in order to provide an objective comparison of the compression performance between the two algorithms.

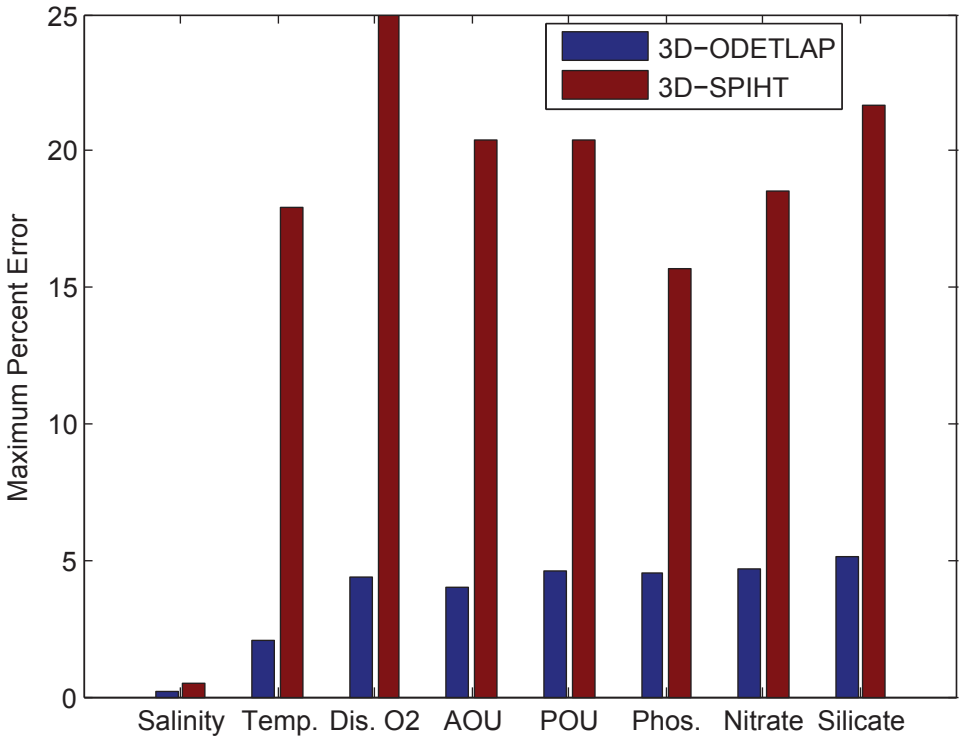

Figure 5.2: Maximum percentage error comparison between 3D-ODETLAP and 3D-SPIHT under same mean percentage error.

The quality of the compression is measured as above in terms of the mean and max percentage error over the range of the current dataset. The results are given in Table 5.4. Please note that we use bzip2 to further compress our results. Because 3D-SPIHT can't compress its results using bzip2 further, the comparison is fair. From the Table 5.4, we can see that while intentionally keeping mean percentage

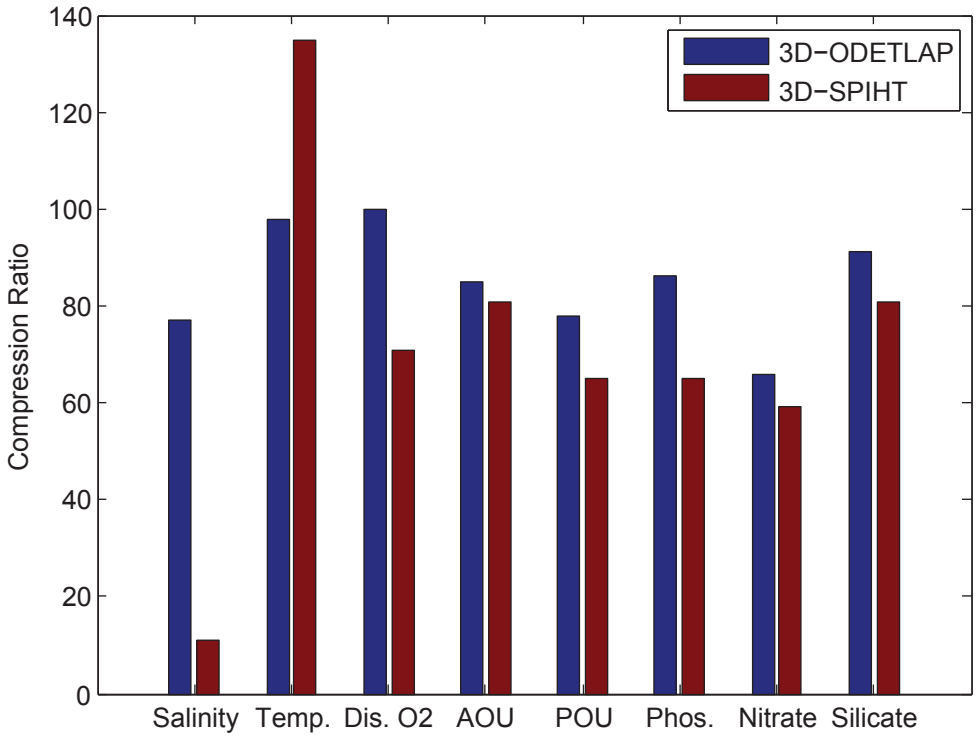

Figure 5.3: Compression ratio comparison between 3D-ODETLAP and 3D-SPIHT under same mean percentage error.

error at approximately 0.15% for salinity dataset and 1% for the seven other datasets, 3D-ODETLAP generally produces a better compression ratio on all eight different datasets and even better maximum percentage error. Figure 5.2 and Figure 5.3 show the advantage of 3D-ODETLAP in terms of maximum error and compression ratio respectively in a clear manner. In fact, we can specify the maximum max error when running 3D-ODETLAP to meet the special needs of some applications, while the 3D-SPIHT compression scheme cannot.

We can see from Table 5.4 that 3D-ODETLAP performs exceptionally well on Salinity dataset when compared with 3D-SPIHT. 3D-ODETLAP has a compression ratio 700% that of 3D-SPIHT's. This is mainly due to the intrinsic data distribution of different datasets. In Figure 5.6, almost 90% of salinity dataset's scalar value fall into the range of 40 to 45 PPS. In contrast, Percentage Oxygen Saturation dataset's scalar value distributes more evenly in its data range as shown in Figure

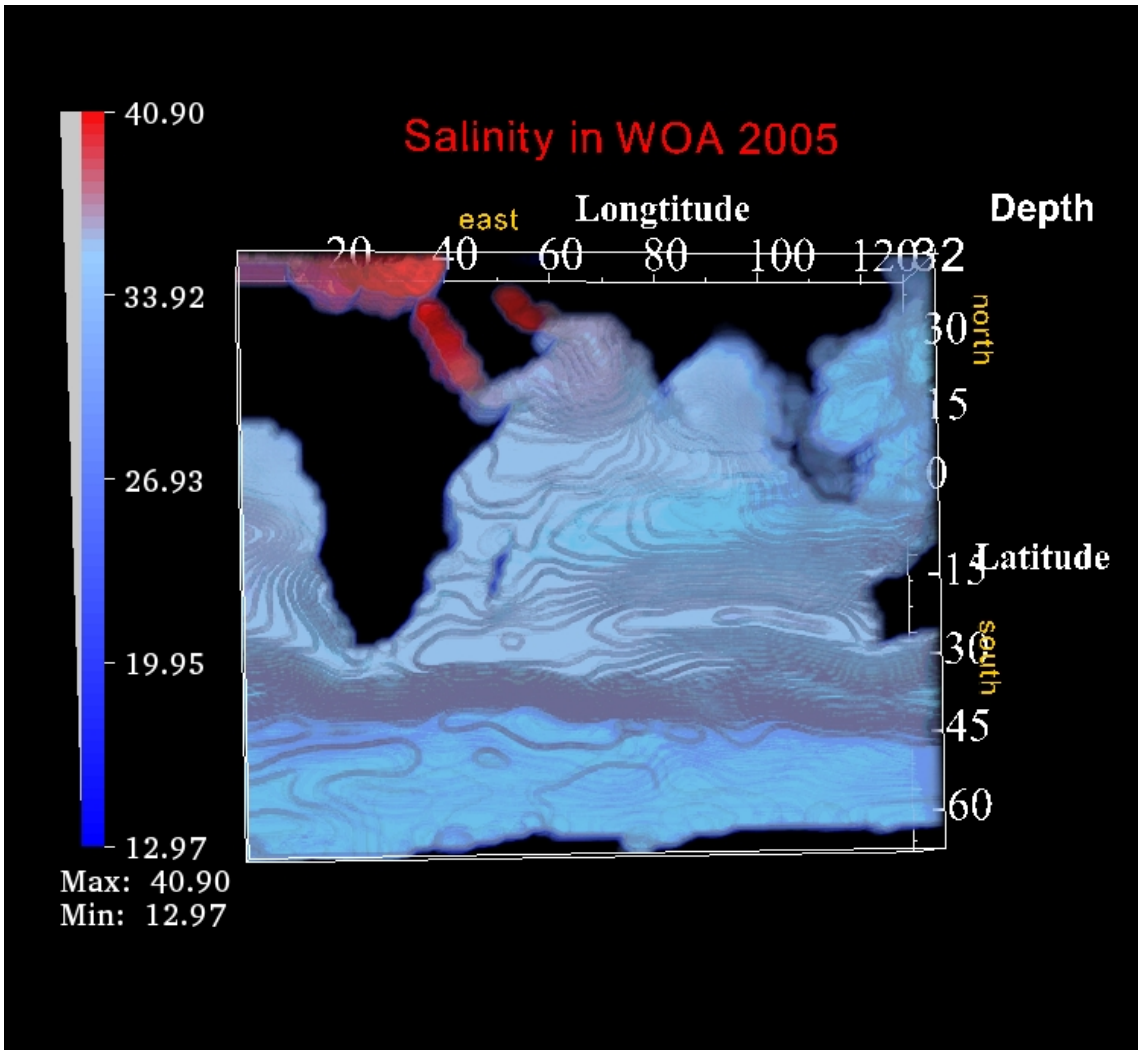

Figure 5.4: Rendering of partial Salinity dataset used in Table 5.4

5.6. Figure 5.4 and Figure 5.5 show their visual difference. Because 3D-ODETLAP approximates the whole marine dataset by selecting certain points to "represent" their neighboring ones, a dataset of almost "flat", like salinity data, will require much fewer points to reconstruct from and thus will have better a compression ratio while keeping errors the same with other methods. This feature of 3D-ODETLAP is especially valuable when we consider that lots of 3D marine, or even environmental datasets have a distribution with similar salinity in WOA 2005. Nevertheless, 3D-SPIHT is designed to be a general purpose compression method and can't take advantage of this special feature of marine datasets.

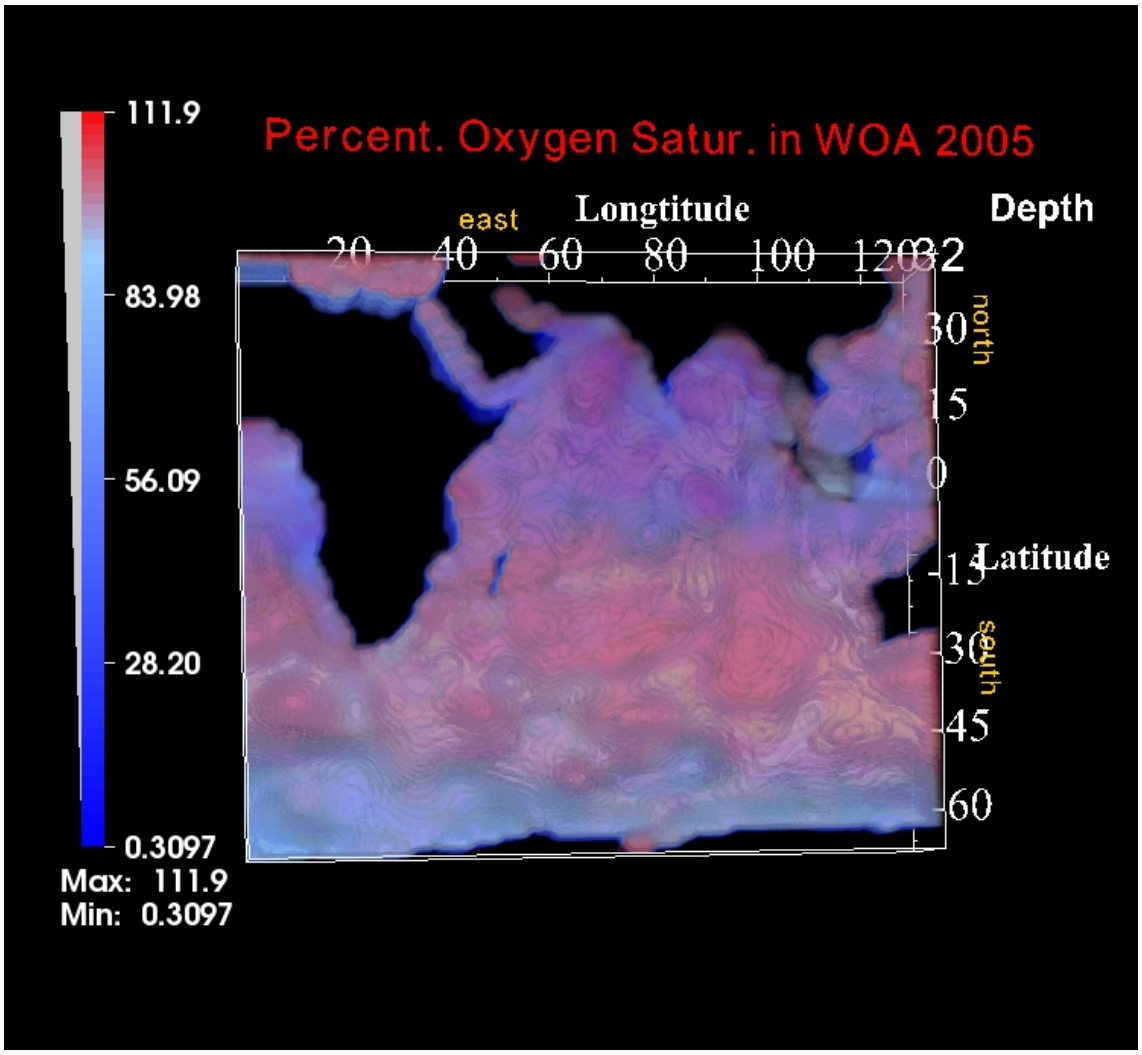

Figure 5.5: Rendering of partial Percent Oxygen Saturation dataset used in Table 5.4.

5.2 Preliminary 5D-ODETLAP

5.2.1 Results and Analysis

At this time, we use *Dividing into Boxes* method to divide this dataset into $4 \times 4 \times 2$ sub-boxes of size $12 \times 12 \times 12 \times 1$. We also use the simple RLE and prefix-coding method from 3D-ODETLAP to gain a preliminary result of 5D-ODETLAP. Our experiment of 5D-ODETLAP is also based on WOA 2005 data. In our experiment for 5D-ODETLAP, we derive four $48 \times 48 \times 24 \times 12$ 4D datasets for fields including *temperature, A.O.U., percentage oxygen saturation* and *dissolved oxygen*. So this derived test dataset has dimension of $48 \times 48 \times 24 \times 12$.

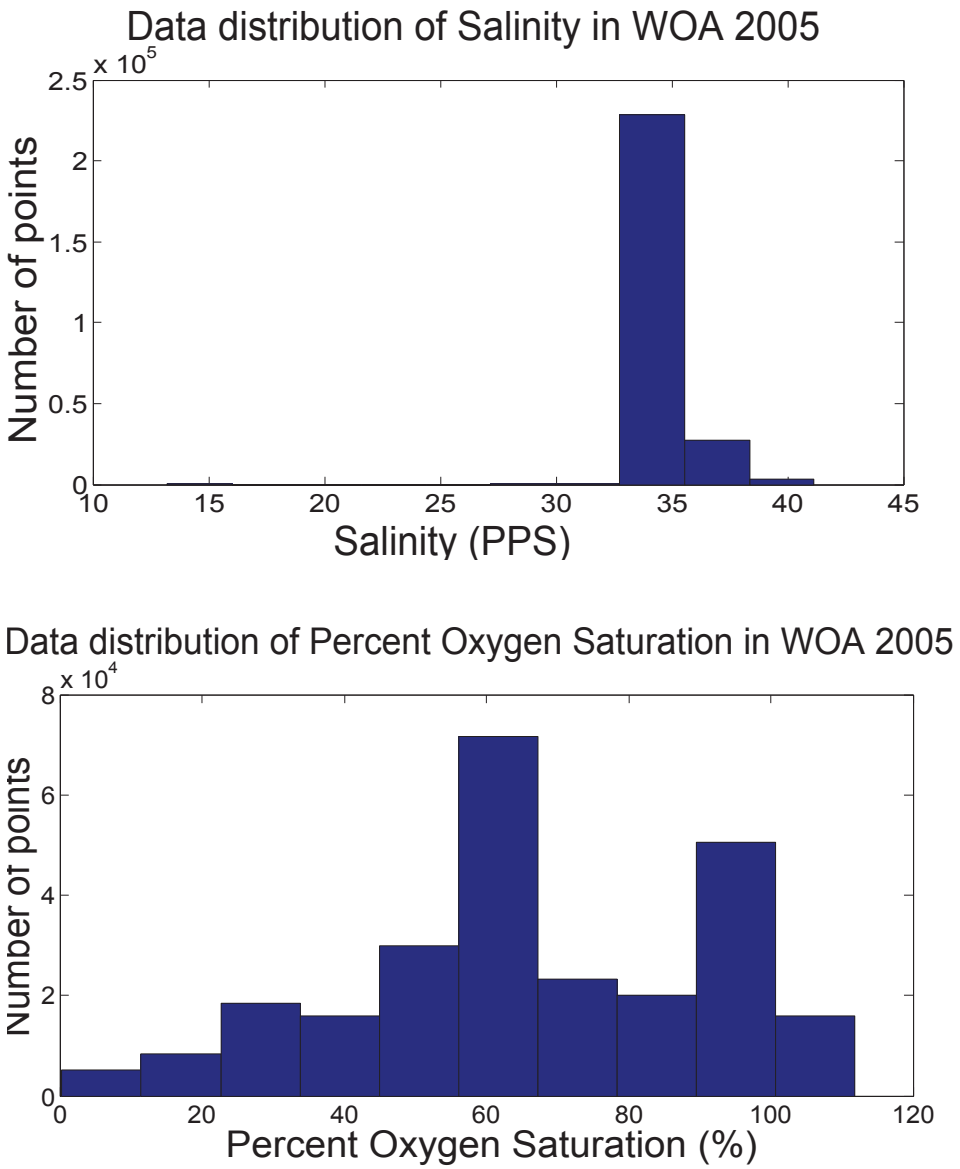

Figure 5.6: Histogram of Salinity and Percentage Oxygen Saturation datasets used in Table 5.6

Table 5.5 shows information about the raw 5D data from WOA 2005. Besides the parameter R and the initial point selection strategy, other parameters can also affect the effectiveness of our method. These are points added at each iteration and size of forbidden zone. Even though their influence on the performance of 5D-ODETLAP are relatively trivial, we still use the best parameters found in our experiments, aiming at achieving high compression ratio while keeping the errors beneath a tolerant level.

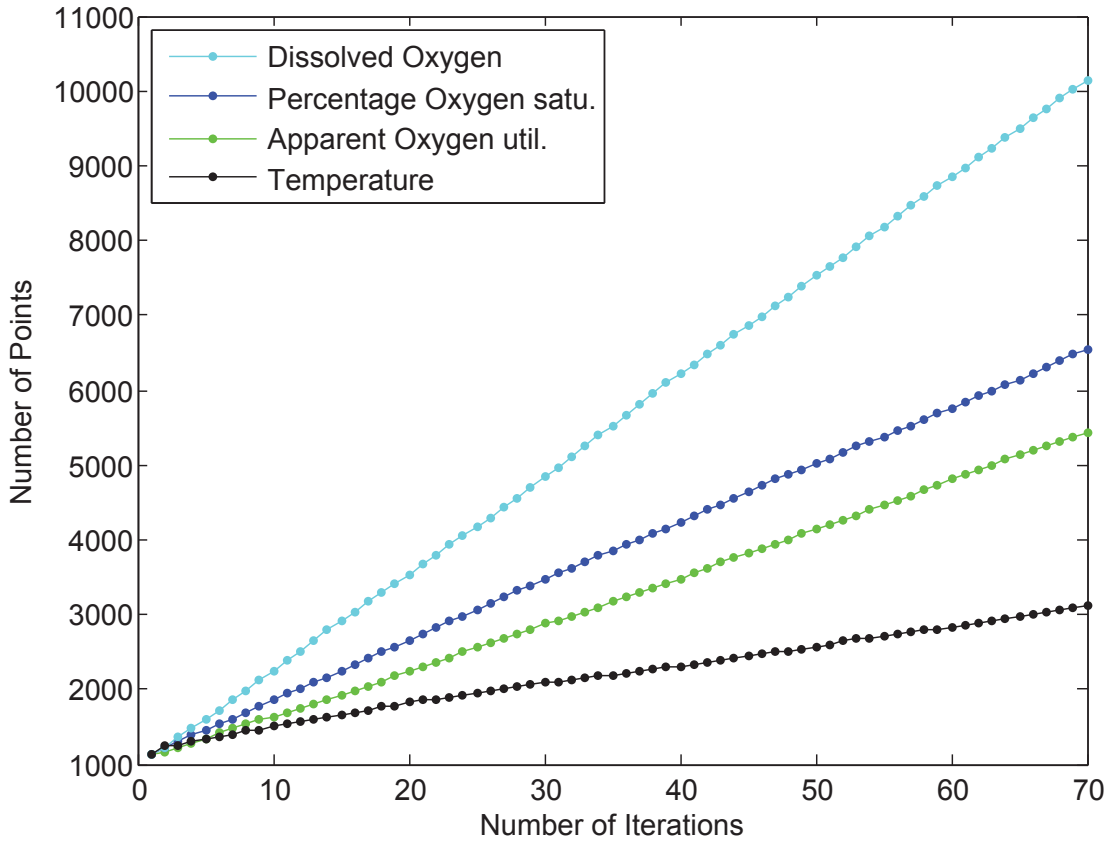

Figure 5.7: Number of points added to each 4D dataset within the 5D dataas, as the iterative process proceeds. The size of each 4D dataset is $48 \times 48 \times 24 \times 12$**, and the size of the test 5D dataset is** $48 \times 48 \times 24 \times 12 \times 4$**.**

Figure 5.7 shows the number of points increases in each 4D dataset as we add 300 points with the largest errors in 5D space. The derived test data have size of $48 \times 48 \times 24 \times 12 \times 4$. We demonstrate the iterative process of 5D-ODETLAP in each 4D dataset. It can be seen from this figure that during each iteration, the points added in each 4D dataset are not equal. Because each 4D data has a different data distribution, it's possible that the approximation of one 4D dataset is more likely to have more erroneous points than others. Figure 5.7 clearly illustrates this possibility. Consequently, since the number of points added at each 4D dataset are different, they tend to have different mean and max percentage errors. Figures 5.8 and 5.9 show that as the points are added, each 4D datasets' mean and maximum percentage error decrease at different speed, and priority is given to the datasets

with most large errors when adding points.

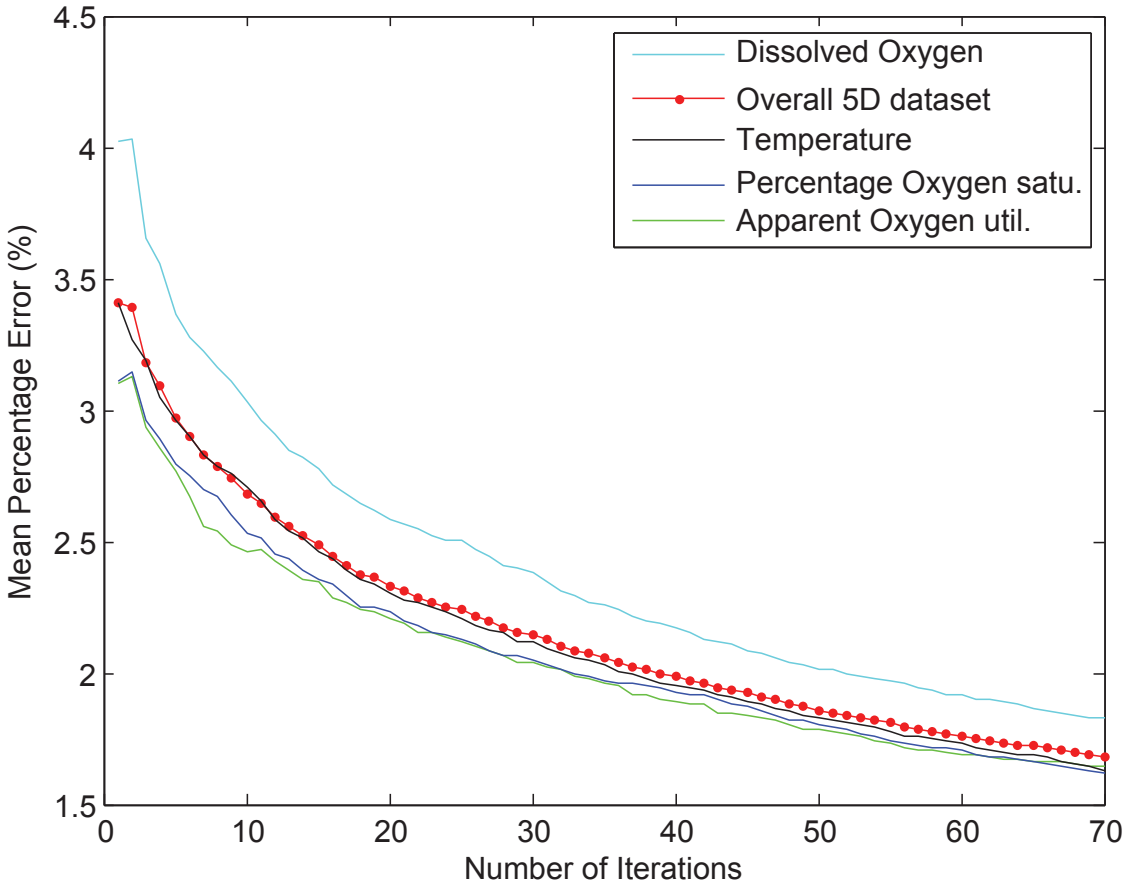

Figure 5.8: Decrease of mean percentage error of each 4D dataset within the 5D dataset and the overall 5D dataset, which is the average mean percentage error of all 4D datasets. Test data is the same as that in Figure 5.7.

5.2.2 Comparison with 3D-SPIHT

We have illustrated the iterative process of compressing a $48 \times 48 \times 24 \times 12 \times 4$ 5D dataset above. In our comparison with the 3D-SPIHT method, we use the same derived dataset with each point caring a single-precision value, stored in 4 bytes each and resulting in a total size of 10.13Mb. We also apply 3D-SPIHT on the same dataset in our experiment in order to provide an objective comparison of the compression performance between the two algorithms. The idea is to compress every 3D block of this 5D data and sum up the size of all compressed 3D blocks. Since the

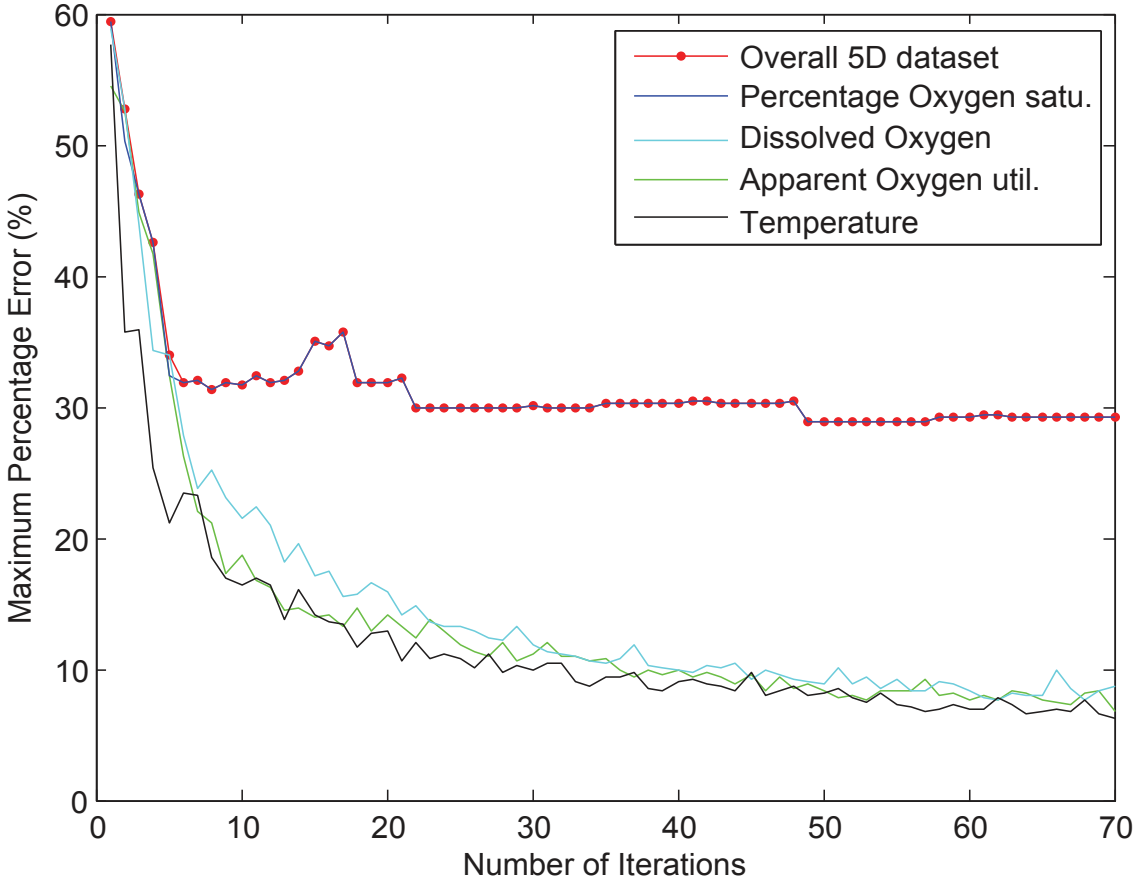

Figure 5.9: Decrease of maximum percentage error of each 4D dataset within the 5D dataset and the overall 5D dataset, which is the maximum percentage error of all 4D datasets. Test data is the same as that in Figure 5.7.

first three dimensions of this 5D data is $48 \times 48 \times 24$, the length of each dimension is not an integer power of 2. In order to apply 3D-SPIHT on this 3D block, zero padding is used to extend 3D block from $48 \times 48 \times 24$ to $64 \times 64 \times 32$.

We can see from Table 5.6 that 5D-ODETLAP outperforms 3D-SPIHT in general. Since 3D-SPIHT is a wavelet-based method, a fixed compression ratio or error can't be set. We actually use binary search to keep the compression ratio and error rate within certain range. First, with approximately same mean percentage error in overall 5D dataset and in every individual 4D dataset, 5D-ODETLAP has a high compression ratio of 110:1 as shown in the upper part of Table 5.6. But the compression ratio of 3D-SPIHT in this case is only 55:1, which is about half

as much as that of our method. Furthermore, the maximum percentage errors of 5D-ODETLAP in overall 5D dataset and in every individual 4D dataset are a lot smaller than the ones from 3D-SPIHT. In addition, our method provides users the flexibility to set a desired maximum or mean percentage error before compression according to their requirements, while 3D-SPIHT doesn't.

Secondly, in the lower part of Table 5.6, if we force the maximum percentage errors of 3D-SPIHT in overall 5D dataset and every individual 4D dataset to be approximately the same as, or even a little worse than, the ones from 5D-ODETLAP results, we can see that 5D-ODETLAP has a compression ratio of 110:1, which is almost four times as much as that of 3D-SPIHT. Since we are proposing a lossy compression method, the maximum error is important because we need to guarantee, in a variety of applications, the compression's error is under a desirable limit according to specific requirements of each individual application. So if we take this into consideration, 5D-ODETLAP is much better than 3D-SPIHT as shown in Table 5.6.

5.3 Optimized 5D-ODETLAP Results and Comparison

With the CUDA-based linear solver, the *Dividing into Box* method is abandoned since the dramatic improvement in the solver's speed enables 5D-ODETLAP to compress the 5D dataset as a whole efficiently. Also, the optimization of 5D-ODETLAP includes a specially-designed further encoding part for the selected sample points subset. Our test of optimized 5D-ODETLAP is based on two real world geospatial 5D datasets — World Ocean Atlas 2005 (WOA05) and WOA 2009 (WOA09), from NODC (the National Oceanographic Data Center). WOA05 and WOA09 both contain a set of objectively analyzed (1 degree grid) climatological fields of in situ *temperature, salinity, dissolved oxygen, Apparent Oxygen Utilization (AOU), percent oxygen saturation, phosphate, silicate,* and *nitrate* at standard depth levels for annual, seasonal, and monthly compositing periods for the global ocean. It serves as an example of high dimensional geospatial datasets. The monthly data we derive from WOA05 and WOA09 is $180 \times 360 \times 24 \times 12 \times 4$, which stores the *temperature, AOU, percentage oxygen saturation* and *dissolved oxygen* in a 12 months period

for the global ocean. In our experiments, we geographically divide these two datasets into 8 different datasets with each one of size $90 \times 180 \times 24 \times 12 \times 4$. They are named WOA05_1, WOA05_2, WOA05_3, WOA05_4, WOA09_1, WOA09_2, WOA09_3 and WOA09_4. In this way, we could test the robustness of our 5D-ODETLAP method on 8 distinct datasets. Table 5.7 shows the data properties of WOA09_1.

Since 3D-SPIHT and JPEG 2000 are among the most popular lossy compression methods, they are an excellent test for 5D-ODETLAP. We assume each value is stored as a single precision float, but the actual storage of the data is not so precise. NODC stores each value in a seven digit real number with 4 places to the right of the decimal. So the compression ratio comes in part from the truncation of digits for storage, but this does not affect the comparison between different compression methods. For a fair comparison, we use binary search in 3D-SPIHT to find a suitable bit rate to produce results with the same mean percentage error in one test and the same maximum percentage error in another case as that of 5D-ODETLAP. The $90 \times 180 \times 24 \times 12 \times 4$ 5D dataset contains 12×4 3D datasets of size $90 \times 180 \times 24$. We apply 3D-SPIHT on each of these 3D datasets and measure the error together as a 5D reconstruction. Similarly, we use binary search to find a compression ratio in JPEG 2000 to obtain the the same error in both cases. We also apply JPEG 2000 on each 90×180 2D dataset of the overall $24 \times 12 \times 4$ number of 2D datasets. In addition, since JPEG 2000 only takes input of unsigned 1, 8 or 16 bits integers, we first use uniform quantization to quantize each input 2D dataset to be stored as 16 bits unsigned integer. All the three methods now have the same mean or maximum percentage error on all the 8 test datasets, so the comparison of their compression ratios is fair.

Table 5.8 shows the comparison result between the three methods when their compressed files have the same mean percentage error. First, the maximum percentage error of compressed file from 5D-ODETLAP is much smaller than that of both JPEG 2000 and 3D-SPIHT. This advantage of the 5D-ODETLAP method is credited to the iterative greedy sampling process, since it eliminates the points with the largest error at each iteration by adding them into the sample set. The JPEG 2000 and 3D-SPIHT methods do not have this adaptability in compression process,

thus produce much large maximum percentage error. Second, the compression ratio of the 5D-ODETLAP method is generally 4.24-9.8 times as large as that of the JPEG 2000 method and 1.13-3.75 times as large as that of the 3D-SPIHT method.

Similarly, if we force the maximum percentage error to be the same in the results from all three methods in Table 5.9, 5D-ODETLAP's compression ratio is even larger than that in the above fixed mean percentage error case. The reason for using this comparison is that maximum error is sometimes essential. For example, if a user needs a compressed file with a guaranteed error of no more than 10%, the compression ratio of both JPEG 2000 and 3D-SPIHT will be quite undesirable, since the mean error of the compressed files will be unnecessarily small. Comparatively, the 5D-ODETLAP's iterative sampling process ensures that the points with largest error are always selected, which reduces the maximum error at each step.

In our implementation, we use binary search to try to fix the maximum percentage error of compressed files from JPEG 2000 and 3D-SPIHT to be the same with that from 5D-ODETLAP. We fail to achieve precisely this because the maximum percentage error is much more unstable than the mean percentage error. So, we use the *cosine similarity* function [56] to measure the similarity between the three maximum percentage error vectors from the three compression methods. The result of the *cosine similarity* function is 1 when the two vectors are identical and 0 when they are opposite. The result of the measurement of maximum percentage error from JPEG 2000 and 3D-SPIHT are 0.9993 and 0.9954 respectively, which means they are almost identical.

	3D-ODETLAP			
Variable	Mean Error(%)	Max Error(%)	Compressed File(bytes)	Compression Ratio
Salinity	0.0532	0.2174	27,377	77:1
Temperature	0.4993	2.0673	21,508	98:1
Dissolved Oxygen	0.9993	4.4145	21,040	100:1
Apparent Oxygen Utilization	0.9999	4.0170	24,775	85:1
Percent Oxygen Saturation	0.9985	4.5672	26,743	78:1
Phosphate	0.9993	4.5241	24,493	86:1
Nitrate	1.0242	4.6946	31,905	66:1
Silicate	0.9996	5.1437	23,076	91:1
	3D-SPIHT			
Variable	Mean Error(%)	Max Error(%)	Compressed File(bytes)	Compression Ratio
Salinity	0.0530	0.4946	190,650	11:1
Temperature	0.50	17.91	15,534	135:1
Dissolved Oxygen	1.002	24.9965	29,537	71:1
Apparent Oxygen Utilization	0.9991	20.3609	25,891	81:1
Percent Oxygen Saturation	0.9969	20.3610	32,264	65:1
Phospha.	0.99784	15.6922	32,264	65:1
Nitrate	1.0006	18.5360	35,545	59:1
Silicate	1.0018	21.6457	25,891	81:1

Table 5.4: Comprehensive compression performance comparison between 3D-ODETLAP and 3D-SPIHT. Due to the data size limitations of 3D-SPIHT, we extract eight $128 \times 128 \times 32$ 3D data matrices and apply both methods on them. Mean Percentage Errors are kept approximately the same for comparison purposes. The size of each dataset is 2Mb with each point uses 4 bytes. For example, compressed Salinity dataset file size by 3D-ODETLAP is 27,377 bytes and thus has a compression ratio of 77:1.

Variable	Unit	Data Range	Size(MB)
Temperature	$^\circ$	[-2.10, 33.23]	71.20
Dissolved oxygen	$\mathrm{ml}l^{-1}$	[0, 11.80]	71.20
Apparent oxygen utilization	$\mathrm{ml}l^{-1}$	[-4.11, 8.42]	71.20
Percent oxygen saturation	%	[0, 158.57]	71.20

Table 5.5: Monthly objectively analyzed climatology datasets [4, 16, 35] on 24 standard depth levels on four variables before compression. Each point uses 4 byte to store its value in single precision; thus total size of each 4D dataset is 71.20MB, which forms a 5D dataset 284.80MB totaling.

	Fixed Mean Percentage Error	
Method	5D-ODETLAP	3D-SPIHT
Mean Err(%)	1.6803 (1.6408, 1.6215, 1.8281, 1.6310)	1.6788 (1.6399, 1.6230, 1.8258, 1.6263)
Max Err(%)	29.2318 (6.7723, 29.2318, 8.7549, 6.2170)	32.7897 (31.0172, 26.3545, 29.1149, 32.7897)
Compressed Size(byte)	96195	192204
Compression Ratio	110.37 : 1	55.24 : 1
	Fixed Maximum Percentage Error	
Method	5D-ODETLAP	3D-SPIHT
Max Err(%)	29.2318 (6.7723, 29.2318, 8.7549, 6.2170)	32.3055 (8.2964, 32.3055, 9.9796, 6.7117)
Mean Err(%)	1.6803 (1.6408, 1.6215, 1.8281, 1.6310)	0.9637 (0.6104, 1.7630, 0.8855, 0.5962)
Compressed Size(Kb)	96195	389495
Compression Ratio	110.37 : 1	27.26 : 1

Table 5.6: Performance comparison between 5D-ODETLAP and 3D-SPIHT on the derived dataset, which is the same with the Data2 dataset in Table 3.2. The upper part of this table shows the results with approximately same mean percentage error of both methods; the lower part shows results with approximately same maximum percentage error. Uniformly random initial selection as in Section 3.2.5 is used for 5D-ODETLAP. The four values within brackets in the rows "Mean Err" and "Max Err" represent respectively values of each 4D dataset, including *apparent oxygen utilization, percentage oxygen saturation, dissolved oxygen* and *temperature*. Mean error is the average of all four 4D datasets' mean percentage error and similarly, Max error is the maximum of all four 4D datasets' maximum percentage error. Each point uses 4 bytes to store its value in single precision; thus total size of each dataset is 10.13MB.

Variable	Unit	Data Range	Size(MB)
Apparent oxygen utilization	ml^{-1}	[-2.29, 6.57]	17.80
Percent oxygen saturation	%	[0, 131.42]	17.80
Dissolved oxygen	ml^{-1}	[0.05, 11.10]	17.80
Temperature	$°C$	[-2.10, 31.18]	17.80

Table 5.7: Data properties of WOA09_1 dataset, which is one of the eight test datasets. Each point uses 4 byte to store its value in single precision; thus total size of each 4D dataset is 17.80MB, which forms a 5D dataset 71.20MB totaling.

Dataset	Fixed Mean Err (%)	Max Err of JPEG 2000 (%)	Max Err of 3D-SPIHT (%)	Max Err of 5D-ODETLAP (%)
woa05_1	1.42	44.83	66.46	10.41
woa05_2	1.48	49.33	59.18	9.35
woa05_3	1.47	65.56	80.23	8.94
woa05_4	1.56	67.56	74.14	10.81
woa09_1	1.46	48.13	68.18	9.021
woa09_2	1.49	51.21	62.15	8.77
woa09_3	1.54	75.00	79.35	11.13
woa09_4	1.58	65.55	71.50	11.58

Dataset	**Ratio** $(\frac{5D-ODETLAP}{JPEG2000})$	**Ratio** $(\frac{5D-ODETLAP}{3D-SPIHT})$
woa05_1	8.16	2.40
woa05_2	9.56	3.61
woa05_3	4.24	1.13
woa05_4	8.57	2.41
woa09_1	8.18	2.41
woa09_2	9.80	3.75
woa09_3	4.24	1.14
woa09_4	8.58	2.42

Table 5.8: Top: Maximum percentage error of 5D-ODETLAP, JPEG 2000 and 3D-SPIHT with the same mean percentage error on eight different datasets. Bottom: The two column indicate the ratio of 5D-ODETLAP's compression ratio over the compression ratio of JPEG 2000 and 3D-SPIHT respectively.

Dataset	Fixed Max Err (%)	Mean Err of JPEG 2000 (%)	Mean Err of 3D-SPIHT (%)	Mean Err of 5D-ODETLAP (%)
woa05_1	10.41	0.52	0.37	1.42
woa05_2	9.35	0.32	0.35	1.48
woa05_3	8.94	0.26	0.28	1.47
woa05_4	10.81	0.37	0.34	1.56
woa09_1	9.02	0.39	0.29	1.46
woa09_2	8.77	0.31	0.36	1.49
woa09_3	11.13	0.36	0.31	1.54
woa09_4	11.58	0.39	0.34	1.58

Dataset	**Ratio** $(\frac{5D-ODETLAP}{JPEG2000})$	**Ratio** $(\frac{5D-ODETLAP}{3D-SPIHT})$
woa05_1	16.33	11.51
woa05_2	24.02	15.74
woa05_3	13.84	9.47
woa05_4	22.23	14.91
woa09_1	18.97	13.87
woa09_2	24.43	15.51
woa09_3	12.05	8.91
woa09_4	21.92	15.22

Table 5.9: Top: Mean percentage error of 5D-ODETLAP, JPEG 2000 and 3D-SPIHT with the same maximum percentage error on eight different datasets. Bottom: The two column indicate the ratio of 5D-ODETLAP's compression ratio over the compression ratio of JPEG 2000 and 3D-SPIHT respectively.

6. CONCLUSION

In this thesis, we present a novel high dimensional geospatial data compression framework, which focuses on high dimensional data compression and is accelerated by CUDA-based linear solver to exploit the parallelism in modern GPUs. A specially designed further encoding scheme for the selected points set is also shown as part of the HD-ODETLAP framework.

Original 5D dataset is iteratively sampled according to the reconstruction from sampled point set by solving an over-determined linear system. This iterative process fully exploits the information redundancy in all the dimensions within this 5D dataset. When decompression, the resulting selected points set could be used to compute the original 5D dataset with a user-specified error. To further improve the compression ratio, the selected points set is encoded using a specially designed coding method based on traditional general compression methods.

Moreover, solving the over-determined linear system is time-consuming. So we incorporated a CUDA-based conjugate gradient linear solver into the HD-ODETLAP framework to utilize the parallelism among the modern GPUs. We have interfaced this linear solver with Matlab to maximize programming efficiency and to minimize data transfer overhead.

The test of HD-ODETLAP is based on eight distinct 5D datasets from WOA 2005 and WOA 2009, which store environmental information such as temperature, salinity, etc. Current popular 3D compression methods, such as JPEG 2000 and SPIHT, extend their 2D compression to 3D. These methods may serve well for general purpose data compression, but for data that preserves features in a specific field, like WOA 2005 in marine study, they lack the flexibility to adjust themselves to the field. By contrast, the HD-ODETLAP method has great adaptability to enable itself to find a good set of parameters to meet the needs for compressing data within the geospatial field. Our technique efficiently exploits spatial and temporal redundancies in high dimensional geospatial data to achieve a high compression ratio on our test data.

Moreover, by applying wavelet-based 3D-SPIHT and JPEG 2000 on the same test datasets, we find that HD-ODETLAP can achieve a much higher compression ratio while intentionally keeping the mean percentage error on all respective datasets the same. We can also obtain a much smaller maximum percentage error in our comparison with them.

7. FUTURE WORK

HD-ODETLAP has demonstrated its effectiveness in high dimensional geospatial data compression in this thesis. But its potential is far beyond this field because of its mathematical generality and the increasing demand for compression in other fields. In fact, HD-ODETLAP provides a general framework of high dimensional data compression for researchers in a wide range of topics, which have difficulties in massive data storage or transmitting. With proper parameters selection, HD-ODETLAP has the ability to approximate various kinds of high dimensional data including those widely used medical 4D FMRI data. This feature may enable doctors and researcher to conduct human brain interaction analysis with limited data acquired from medical devices. For example, a FMRI scan of one patient's brain in certain time range may have some data missing. But the doctors need a reasonable 4D model to diagnose and this 4D data may be too large to store or transmit. HD-ODETLAP can use all the data obtained so far to make a reasonable smooth reconstruction of the whole brain, fill in those missing data and compress it effectively.

Similarly, in Computational Fluid Dynamics (CFD) field, a simple 60 seconds simulation could produce Gigabytes of 5D data. These data are often sparsely uniformly sampled and compressed for storage because of its large size. HD-ODETLAP could be applied on this kind of 5D datasets since they are 5D in nature and require aggressive compression, which match the unique features of the HD-ODETLAP method.

One useful extension of current HD-ODETLAP method lies in further exploiting the correlation between the 4D datasets representing distinct variables. In computing a 5D approximation from sampled points, we construct linear systems for each 4D dataset and then merge them back. And we enable 5D compression in the sampling process. In this way, the autocorrelation between these 4D datasets is partially ignored. A valid model indicating the autocorrelation between these 4D datasets could further improve the compression ratio. Building this model may

require quantitative knowledge about the dataset, and weighting between the reduction in compressed file size and the model's size could lead to future research.

In addition, HD-ODETLAP is completely parallelizable as shown in our preliminary effort of 5D-ODETLAP in Section 5.2. Jake Stookey [54] showed that the original ODETLAP could be parallelized for IBM Blue Gene cluster to compress huge datasets. Similarly, our HD-ODETLAP could be parallelized for large cluster using that idea. This will enable HD-ODETLAP to compress even larger datasets faster than it does currently.

LITERATURE CITED

[1] Adobe Developers Assoc., "TIFF Revision 6.0 Final - June 3, 1992," http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf, Date Last Accessed 10/06/2011.

[2] AMD Corp., "ATI Stream Technology," http://developer.amd.com/gpu/AMDAPPSDK-/Pages/default.aspx, Date Last Accessed 2/14/2011.

[3] K. Anagnostou, T. Atherton, and A. Waterfall, "4D volume rendering with the Shear Warp factorisation," in *Proceedings of the 2000 IEEE symposium on Volume visualization*. ACM, 2000, p. 137.

[4] J. I. Antonov, R. A. Locarnini, T. P. Boyer, A. V. Mishonov, and H. E. Garcia, "World ocean atlas 2005, volume 2: Salinity," p. 182, 2006.

[5] J. I. Antonov, D. Seidov, T. P. Boyer, R. A. Locarnini, A. V. Mishonov, H. E. Garcia, O. K. Baranova, M. M. Zweng, and D. R. Johnson, "World ocean atlas 2009, volume 2: Salinity," p. 184, 2010.

[6] P. Baumann, S. Feyzabadi, and C. Jucovschi, "Putting Pixels in Place: A Storage Layout Language for Scientific Data," *Data Mining Workshops, International Conference on*, vol. 0, pp. 194–201, 2010.

[7] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: ACM, 2009, pp. 18:1–18:11. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654078

[8] ——, "CUSP: Generic Parallel Algorithms for Sparse Matrix and Graph Computations," http://cusp-library.googlecode.com, Date Last Accessed 7/15/2011, version 0.1.0.

[9] C. Binnig, S. Hildenbrand, and F. Färber, "Dictionary-based order-preserving string compression for main memory column stores," in *Proceedings of the 35th SIGMOD international conference on Management of data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 283–296. [Online]. Available: http://doi.acm.org/10.1145/1559845.1559877

[10] J. T. Bjøke and S. Nilsen, "Efficient representation of digital terrain models: compression and spatial decorrelation techniques," *Computers & Geosciences*, vol. 28, no. 4, pp. 433–445, 2002.

[11] L. Buatois, G. Caumon, and B. Lévy, "Concurrent Number Cruncher : An Efficient Sparse Linear Solver on the GPU," ser. Lecture Notes in Computer Science, vol. 4782, University of Houston. Houston États-Unis: Springer Berlin / Heidelberg, 2007, pp. 358–371. [Online]. Available: http://hal.inria.fr/inria-00186833/PDF/HPCC_number_cruncher.pdf

[12] W. R. Franklin, "Applications of geometry," in *Handbook of Discrete and Combinatorial Mathematics*, K. H. Rosen, Ed. CRC Press, 2000, ch. 13.8, pp. 867–888.

[13] W. R. Franklin, M. Inanc, and Z. Xie, "Two novel surface representation techniques," in *Autocarto 2006*. Vancouver Washington: Cartography and Geographic Information Society, 25-28 June 2006.

[14] W. R. Franklin, Z. Xie, E. Lau, and Y. Li, "Algorithms for terrain and bathymetric sensor data," in *ICA Workshop on Advances in Sensors and Algorithms for Topographic and Thematic Mapping*. Orlando, Florida: The International Cartographic Association (ICA) Commission on Mapping from Satellite Imagery, 19 Nov. 2010.

[15] H. E. Garcia, R. A. Locarnini, T. P. Boyer, and J. I. Antonov, "World ocean atlas 2005, volume 3: Dissolved oxygen, apparent oxygen utilization, and oxygen saturation," p. 342, 2006.

[16] ——, "World ocean atlas 2005, volume 4: Nutrients (phosphate, nitrate, silicate)," p. 396, 2006.

[17] H. E. Garcia, R. A. Locarnini, T. P. Boyer, J. I. Antonov, O. K. Baranova, M. M. Zweng, and D. R. Johnson, "World ocean atlas 2009, volume 3: Dissolved oxygen, apparent oxygen utilization, and oxygen saturation," p. 344, 2010.

[18] H. E. Garcia, R. A. Locarnini, T. P. Boyer, J. I. Antonov, M. M. Zweng, O. K. Baranova, and D. R. Johnson, "World ocean atlas 2009, volume 4: Nutrients (phosphate, nitrate, silicate)," p. 398, 2010.

[19] A. Getis, "Spatial autocorrelation," in *Handbook of Applied Spatial Analysis*, M. M. Fischer and A. Getis, Eds. Springer Berlin Heidelberg, 2010, pp. 255–278.

[20] M. F. Goodchild and Y. Shiren, "A hierarchical spatial data structure for global geographic information systems," *CVGIP: Graphical Models and Image Processing*, vol. 54, no. 1, pp. 31–44, 1992.

[21] M. B. Gousie and W. R. Franklin, "Augmenting Grid-based Contours to Improve Thin Plate DEM Generation," *Photogrammetric Engineering & Remote Sensing*, vol. 71, no. 1, pp. 69–79, 2005.

[22] ——, "Converting elevation contours to a grid," in *Eighth International Symposium on Spatial Data Handling*. Vancouver BC Canada: Dept of Geography, Simon Fraser University, Burnaby, BC, Canada, July 1998, pp. 647–656.

[23] J. Hensley, "AMD CTM overview," in *ACM SIGGRAPH 2007 courses*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1281500.1281648

[24] X. Jin, S. Chen, and X. Mao, "Computer-Generated Marbling Textures: A GPU-Based Design System," *Computer Graphics and Applications, IEEE*, vol. 27, no. 2, pp. 78–84, Mar.-Apr. 2007.

[25] B. Khailany, W. J. Dally, S. Rixner, U. J. Kapasi, J. D. Owens, and B. Towles, "Exploring the VLSI Scalability of Stream Processors," *High-Performance Computer Architecture, International Symposium on*, vol. 0, p. 153, 2003.

[26] D. B. Kidner and D. H. Smith, "Advances in the data compression of digital elevation models," *Computers & Geosciences*, vol. 29, no. 8, pp. 985–1002, 2003.

[27] B.-J. Kim and W. A. Pearlman, "An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPIHT)," in *Data Compression Conference, 1997. DCC '97. Proceedings*, Mar. 1997, pp. 251–260.

[28] Y. Kim and W. Pearlman, "Lossless volumetric medical image compression," *Proc. of SPIE, Applications of Digital Image Processing*, vol. 3808, pp. 305–312, 1999.

[29] D. E. Knuth, "Dynamic huffman coding," *Journal of Algorithms*, vol. 6, no. 2, pp. 163–180, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0196677485900367

[30] H. Lalgudi, A. Bilgin, M. Marcellin, and M. Nadar, "Compression of fMRI and ultrasound images using 4D SPIHT," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 2, Sept. 2005, pp. II – 746–9.

[31] T.-Y. Lau, Y. Li, Z. Xie, and W. R. Franklin, "Sea floor bathymetry trackline surface fitting without visible artifacts using ODETLAP," in *17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2009)*, Seattle WA USA, 4-6 Nov. 2009.

[32] Y. Li, T.-Y. Lau, P. Fox, and W. R. Franklin, "5D-ODETLAP: A novel high-dimensional compression method on time-varying geospatial data," in *Geospatial Data and Geovisualization: Environment, Security, and Society, a*

special joint symposium of ISPRS Technical Commission IV & AutoCarto 2010, Orlando, Florida, 15-18 Nov. 2010.

[33] Y. Li, T.-Y. Lau, C. Stuetzle, P. Fox, and W. R. Franklin, "3D oceanographic data compression using 3D-ODETLAP," in *18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2010)*, San Jose, CA, USA, 2-5 Nov. 2010, (PhD Dissertation showcase).

[34] S. Lloyd, "Least squares quantization in PCM," *Information Theory, IEEE Transactions on*, vol. 28, no. 2, pp. 129–137, Mar. 1982.

[35] R. A. Locarnini, A. V. Mishonov, J. I. Antonov, T. P. Boyer, and H. E. Garcia, "World ocean atlas 2005, volume 1: Temperature," p. 182, 2006.

[36] R. A. Locarnini, A. V. Mishonov, J. I. Antonov, T. P. Boyer, H. E. Garcia, O. K. Baranova, M. M. Zweng, and D. R. Johnson, "World ocean atlas 2009, volume 1: Temperature," p. 184, 2010.

[37] D. Luebke, M. Harris, N. Govindaraju, A. Lefohn, M. Houston, J. Owens, M. Segal, M. Papakipos, and I. Buck, "GPGPU: general-purpose computation on graphics hardware," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1188455.1188672

[38] E. B. Lum, K. L. Ma, and J. Clyne, "Texture hardware assisted rendering of time-varying volume data," in *VIS '01: Proceedings of the conference on Visualization '01*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 263–270.

[39] MathWorks Corp., "Matlab - mldivide, mrdivide," http://www.mathworks.com/access/helpdesk/help/techdoc/ref/mldivide.html, Date Last Accessed 7/15/2011.

[40] K. Mehlhorn and S. Näher, *LEDA: a platform for combinatorial and geometric computing*. New York, NY, USA: Cambridge University Press, 1999.

[41] G. Menegaz and J.-P. Thiran, "Lossy to lossless object-based coding of 3-d mri data," *Image Processing, IEEE Transactions on*, vol. 11, no. 9, pp. 1053–1061, Sep. 2002.

[42] J. Mennis, R. Viger, and C. Tomlin, "Cubic Map Algebra Functions for Spatio-Temporal Analysis," *Cartography and Geographic Information Science*, vol. 32, no. 1, pp. 17–33, 2005.

[43] A. Moffat, "Implementing the PPM data compression scheme," *Communications, IEEE Transactions on*, vol. 38, no. 11, pp. 1917–1921, Nov. 1990.

[44] A. Munshi, "OpenCL: Parallel Computing on the GPU and CPU," *SIGGRAPH, Tutorial*, 2008.

[45] M. R. Nelson, "LZW data compression," *Dr. Dobb's J.*, vol. 14, pp. 29–36, Oct. 1989. [Online]. Available: http://portal.acm.org/citation.cfm?id=77102.77104

[46] NVIDIA Corp., "NVIDIA Corporation: Compute Unified Device Architecture Programming Guide," http://developer.nvidia.com/cuda, Date Last Accessed 1/11/2011.

[47] J. D. Owens, B. Khailany, B. Towles, and W. J. Dally, "Comparing Reyes and OpenGL on a stream architecture," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ser. HWWS '02. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2002, pp. 47–56. [Online]. Available: http://portal.acm.org/citation.cfm?id=569046.569053

[48] A. Plaza, J. Plaza, and A. Paz, "Improving the scalability of hyperspectral imaging applications on heterogeneous platforms using adaptive run-time data compression," *Computers & Geosciences*, vol. 36, no. 10, pp. 1283–1291, 2010.

[49] I. E. Richardson, *Front Matter*. John Wiley & Sons, Ltd, 2004, pp. i–xxiv. [Online]. Available: http://dx.doi.org/10.1002/0470869615.fmatter

[50] V. Sanchez, P. Nasiopoulos, and R. Abugharbieh, "Lossless Compression of 4D Medical Images using H.264/AVC," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 2, May 2006, p. II.

[51] J. Seward, "The bzip2 and libbzip2 official home page," http://bzip.org/, Date Last Accessed 7/15/2011.

[52] G. Sewell, *Boundary Value Problems*. John Wiley & Sons, Inc., 2006, pp. 120–173.

[53] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE signal processing magazine*, vol. 18, no. 5, pp. 36–58, 2001.

[54] J. Stookey, Z. Xie, B. Cutler, W. R. Franklin, D. M. Tracy, and M. A. Andrade, "Parallel ODETLAP for terrain compression and reconstruction," in *GIS '08: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, 2008.

[55] D. Sturgill, B. Van Ruitenbeek, and R. Marks, "Image compression and recovery through compressive sampling and particle swarm," in *Systems, Man*

and Cybernetics, 2009. SMC 2009. IEEE International Conference on, Oct. 2009, pp. 1821–1826.

[56] P. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining.* Pearson Addison Wesley, 2006, p. 500.

[57] D. S. Taubman, M. W. Marcellin, and M. Rabbani, "Jpeg2000: Image compression fundamentals, standards and practice," *Journal of Electronic Imaging*, vol. 11, p. 286, 2002.

[58] D. Tracy, "Path planning and slope representation on compressed terrain," Ph.D. dissertation, RENSSELAER POLYTECHNIC INSTITUTE, 2010.

[59] L. Wang, J. Wu, L. Jiao, and S. Zhang, "Three-dimensional medical image compression based on low-complexity RKLT," *Electronics Letters*, vol. 46, no. 6, pp. 395–397, 18 Mar. 2010.

[60] M. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, 2000.

[61] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, pp. 520–540, June 1987. [Online]. Available: http://doi.acm.org/10.1145/214762.214771

[62] Z. Xie, W. R. Franklin, B. Cutler, M. A. Andrade, M. Inanc, and D. M. Tracy, "Surface compression using over-determined Laplacian approximation," in *Proceedings of SPIE Vol. 6697 Advanced Signal Processing Algorithms, Architectures, and Implementations XVII.* San Diego CA: International Society for Optical Engineering, 27 Aug. 2007, paper 6697-15.

[63] Z. Xie, W. R. Franklin, and D. Tracy, "Slope preserving lossy terrain compression," in *18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2010)*, San Jose, CA, USA, 2-5 Nov. 2010, (PhD Dissertation Showcase).

[64] W. Yang, Y. Lu, F. Wu, J. Cai, K. Ngan, and S. Li, "4-D wavelet-based multiview video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 11, pp. 1385–1396, 2006.

[65] L. W. Zevenbergen and C. R. Thorne, "Quantitative analysis of land surface topography," *Earth Surface Processes and Landforms*, vol. 12, no. 1, pp. 47–56, 1987. [Online]. Available: http://dx.doi.org/10.1002/esp.3290120107

[66] G. Ziegler, H. Lensch, M. Magnor, and H.-P. Seidel, "Multi-video compression in texture space using 4D SPIHT," in *Multimedia Signal Processing, 2004 IEEE 6th Workshop on*, Sep.-Oct. 2004, pp. 39–42.

APPENDIX A
Sample CUSP code

```
1
2  #include <cusp/hyb_matrix.h>
3  #include <cusp/gallery/poisson.h>
4  #include <cusp/krylov/cg.h>
5
6  // computation on GPU
7  typedef cusp::device_memory Device;
8
9  int main(void)
10 {
11     // create an empty sparse matrix in HYB format
12     cusp::hyb_matrix<int, float, Device> A;
13
14     // create a 2d Poisson problem on a 10x10 mesh
15     cusp::gallery::poisson5pt(A, 100, 100);
16
17     // allocate memory on GPU for solution vector (x) and right↩
           hand side vector(b)
18   // initial value for (x) set to 0 to start iteration
19     cusp::array1d<float, Device> x(A.num_rows, 0);
20     cusp::array1d<float, Device> b(A.num_rows, 1);
21
22     // Stopping condition:
23     // maximum iterations = 200
24     // tolerance = 1e-6
25     cusp::verbose_monitor<float> monitor(b, 200, 1e-6);
26
```

```
27      // Identity preconditioner
28      cusp::identity_operator<float, Device> M(A.num_rows, A.↩
            num_rows);
29
30      // solve the linear system A*x = b using the Conjugate ↩
            Gradient method
31      cusp::krylov::cg(A, x, b, monitor, M);
32
33      return 0;
34  }
```

APPENDIX B
Sample code in mixed CUSP and MEX style

```
1  #include <cusp/io/matrix_market.h>
2  #include <cusp/coo_matrix.h>
3  #include <cusp/print.h>
4  #include <cusp/krylov/cg.h>
5  #include "mex.h"
6  #include <iostream>
7  using namespace std;
8
9  typedef cusp::device_memory  Device;
10 typedef cusp::host_memory  Host;
11
12 typedef cusp::array1d<float, Device> Array1d;
13 typedef cusp::dia_matrix<int,float,Device> DIA;
14
15 //Mex function entry
16 void mexFunction(int nlhs, mxArray *plhs[],int nrhs, const ↩
       mxArray *prhs[])
17 {
18     if (nrhs != 3){
19         mexErrMsgTxt("Three input argument required");
20         int n = mxGetM(prhs[1]);
21         if (n == 1){
22             mexErrMsgTxt("Transpose the column vector");
23         }
24     }
25     if (nlhs > 1){
26         mexErrMsgTxt("Too many output arguments.");
```

```
27      }
28      if (!mxIsSparse(prhs[0])) {
29          mexErrMsgTxt("First argument must be a sparse array.");
30      }
31
32    DIA A;
33      //read matrix into device
34      read_matrix(prhs[0], A);
35      //read column vector into device
36      int n = mxGetN(prhs[0]);
37      Array1d b(n);
38      read_column(prhs[1],b);
39      Array1d x (n,0);
40
41      //solving the system
42      double *arg = mxGetPr(prhs[2]);
43      cusp::default_monitor<float> monitor(b,(int)(arg[1]), arg↩
            [0]);
44      cusp::identity_operator<float,Device> M(A.num_rows,A.↩
            num_rows);
45    //CUSP conjugate gradient solver
46      cusp::krylov::cg(A,x,b,monitor,M);
47
48      //if solver works
49      if(monitor.converged()){
50          mexPrintf("\tconverged after %d iterations.\n", monitor↩
                .iteration_count());
51          plhs[0] = mxCreateDoubleMatrix(n, 1, mxREAL);
52          double *x_result = mxGetPr(plhs[0]);
53          for(int i =0; i != n; ++i)
54              x_result[i] = x[i];
55      }
56      else//not working
```

```
57      {
58          mexPrintf("\treached iteration limit at %d.\n",monitor.↩
               iteration_limit());
59          plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL);
60          double *x_result = mxGetPr(plhs[0]);
61          x_result[0] = 0.0;
62      }
63
64    return;
65  }
```