Chapter 1
CUDA-Accelerated HD-ODETLAP: Lossy High Dimensional Gridded Data Compression

W. Randolph Franklin, You Li, Tsz-Yam Lau, and Peter Fox[1]

Abstract We present *High-dimensional Overdetermined Laplacian Partial Differential Equations* (HD-ODETLAP), an algorithm and implementation for lossy compression of high-dimensional arrays of data. HD-ODETLAP exploits autocorrelations in the data in any dimension. It also adapts to regions in the data with varying value ranges, resulting in the maximum error being closer to the RMS error. HD-ODETLAP compresses a data array by iteratively selecting a representative set of points from the array. That set of points, efficiently coded, is the compressed dataset. The compressed dataset is uncompressed by solving an overdetermined sparse system of linear equations for an approximation to the original array. HD-ODETLAP uses NVIDIA CUDA called from MATLAB to exploit GPU parallel processing to achieve considerable speedup compared to execution on a CPU. In addition, HD-ODETLAP compresses much better than JPEG2000 and 3D-SPIHT, when fixing either the average or the maximum error. An application is to facilitate storage and transmission of voluminous datasets for better climatological and environmental analysis and prediction.

1.1 Introduction and Background

The research theme of this paper is the use of modern accelerator technologies to address the problem of storing increasing volumes of multidimensional geospatial data using lossy compression. The good compression algorithms needed to maximize the feasible dataset size are quite compute-intensive. Also, the datasets' multi-dimensional structure is not exploited by current compression algorithms. In this paper, we introduce **HD-ODETLAP**, a *High-Dimensional Over-determined Laplacian Partial Differential Equation* compression algorithm and implementation. HD-ODETLAP has various versions, such as 4D-ODETLAP and 5D-ODETLAP.

[1] Rensselaer Polytechnic Institute, Troy, NY, USA, email: mail@wrfranklin.org, liyou.rpi@gmail.com, rpi.laut@gmail.com, pfox@cs.rpi.edu

Their novel technique resides in expressing the problem as an overdetermined sparse system of linear equations.

Our test datasets are the *World Ocean Atlas 2005* and *2009* [14] from the National Oceanographic Data Center (NODC) and the National Geophysical Data Center (NGDC), subsidiaries of the NOAA Atmospheric Administration (NOAA). They contain marine properties *temperature, salinity, nitrate*, and *silicate* over 12 months at 24 standard depths in the ocean. They can be considered to be five-dimensional: *(latitude, longitude, depth, time, property value)*, of size $180 \times 360 \times 24 \times 12 \times 4$. The data is autocorrelated along each dimension: small changes in any coordinate cause small changes in the measured quantity. This applies even to the fifth dimension; the observed variables do, to some extent, vary in concert.

However, current compression methods treating the data as a single-dimensional stream of bytes ignore that correlation. To the extent that HD-ODETLAP can exploit this property, HD-ODETLAP will compress better, and it does on the test data. HD-ODETLAP also adapts to nonhomegeneities in the data, where some local regions may have larger value ranges than other regions. Therefore, an HD-ODETLAP representation with a given RMS error may have a smaller maximum error than do other methods. This is also confirmed by experiment. However, the problem with HD-ODETLAP is its compute-intensiveness, and so, modern accelerator techniques are desirable.

ODETLAP, initially in a 2D version, was developed as part of a project to lossily compress terrain (elevation of the earth's surface above the geoid or assumed sea level). A goal was to facilitate operations such as multi-observer siting, and then path planning to avoid the observers (the *Smugglers and Border Guards Problem*). The initial goal was to interpolate terrain from isolated data points and contour lines. The contour lines may be kidney-bean shaped and might have gaps. Those properties caused problems with earlier interpolation algorithms, such as running straight lines in the eight cardinal directions from the test point to the nearest contour line. In addition, terrain data often have limited precision, and may be mutually inconsistent. However, the interpolated slope should be continuous across lines of data points; that is, a line of data should not cause a visible "kink" in the generated surface. ODETLAP addresses these problems well [4, 5, 7, 11, 18, 22, 23, 25, 26]. Figure 1.1 shows how successfully ODETLAP handles an example designed to be very difficult, with input contours with sharp corners. Nevertheless the interpolated output surface has smooth silhouette edges, and inferred peak inside the innermost contour. All this is achieved with a mean error of 2.7% of the data range (computed on the known points) and a maximum error of 12.9%.

1.2 Compression Basics

Various compression methods have been created for lower dimensional data, such as 3D image sequences [17] and 4D (for example, 3D spatial + temporal) functional magnetic resonance imaging (FMRI) [10]. Among those methods, the wavelet-based

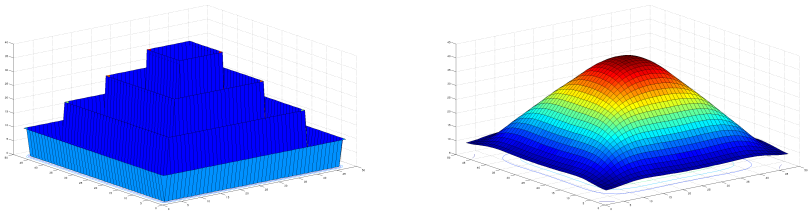

Fig. 1.1 ODETLAP Fitting a Smooth Surface to Nested Square Contours

lossy and lossless ones are the most popular. For example, JPEG 2000 [24] uses irreversible and reversible wavelet transform for its lossy and lossless compression for 2D images. For 3D video data, *Three Dimensional Set Partitioning in Hierarchical Trees* (3D-SPIHT) [9] also bases its compression scheme on 3D wavelet transforms. There are some 4D compression algorithms, such as 4D wavelets [27], run length encoding (RLE) [1] and discrete cosine transform (DCT) [15].

Data compression techniques may be *lossless* or *lossy*. Lossless schemes allow exact reconstruction of the original data but suffer from a low compression ratio. Lossy schemes produce much more compact datasets, at an, often modest, increase, δ, in the dataset's RMS error, ε. If $\delta << \varepsilon$, then the lossy compression has not cost very much accuracy. For example, consider suppose that a uniformly distributed temperature parameter is quantized to integral degrees, so that $\varepsilon = 1/(2\sqrt{3}) \approx 0.3$. $\delta = 0.1$ might allow a much smaller dataset with a small cost in increased RMS error. Experiments on elevation data that trade off error with compressed size are described in [6]. The reader might also experiment by comparing compression quality and size when generating JPEG images. For these reasons, HD-ODETLAP is lossy.

Compression algorithms operate by exploiting redundancies and correlations in the data. Current techniques that compress high dimensional data by compressing 2D or 3D slices separately ignore the correlation between the slices. We don't.

3D compression methods can also be applied on 4D spatial-temporal data, since 4D data could be handled 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 [21] and 4D-SPIHT[28]. These methods fully utilize the spatial and temporal data redundancy in all the dimensions; thus they can achieve a higher compression ratio.

In GIS, there is an increasing awareness of the significance of compression and progressive transmission of high dimensional spatial data. Plaza[20] introduces adaptive run-time data compression of spatial data. Kidner[8] comprehensively 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, [3]. We have published a 3D oceanographic data compression method, [12].

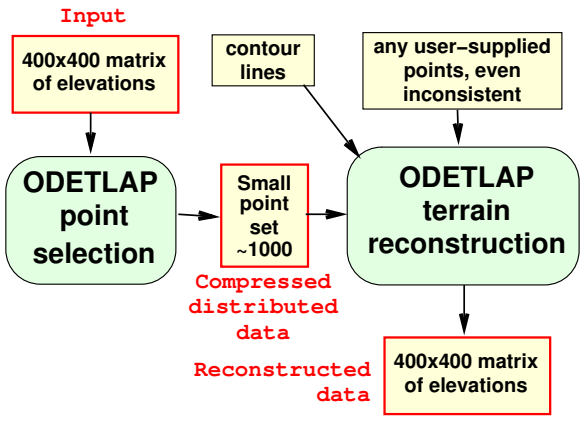

Fig. 1.2 ODETLAP Process

1.3 ODETLAP Definition

We will first present a 2D version of *Five Dimensional Over-Determined Laplacian Partial Differential Equations* (5D-ODETLAP), then extend it. The domain is a 2D array of elevations. The (i, j) entry has the position (x, y) when projected vertically onto the geoid. (x, y) are typically either latitude and longitude, or Universal Transverse Mercator (UTM) coordinates. z is the elevation above the geoid.

ODETLAP has two components. The smaller component is to interpolate from a set of known point elevations to a complete elevation array. The larger component is to lossily compress an elevation array by selecting an important set of points $\mathscr{S}$ from the array. $\mathscr{S}$ is the compressed representation. $\mathscr{S}$ is uncompressed by using ODETLAP to interpolate back to the whole elevation array. The process is illustrated in Figure 1.2.

The interpolation component of 2D-ODETLAP is an extension of the Laplacian PDE

$$\frac{\delta^2 z}{\delta x^2} + \frac{\delta^2 z}{\delta y^2} = 0 \tag{1.1}$$

to an overdetermined linear system. The purpose of this system is to interpolate elevations from a small set of known points to the entire array. This system comprises two types of equations. First, every nonborder point, known or unknown, induces an averaging equation that sets its value to the average of its neighbors:

$$4z_{ij} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i.j+1} \tag{1.2}$$

Second, we add another equation for each known point,

$$z_i = s_i \tag{1.3}$$

where s_i stands for the input known value of the point, and z_i its output computed value. Since ODETLAP is lossy, these values will differ slightly; the fitted surface does not exactly interpolate the known data. This system is over-determined, since there are more equations than unknowns. Being over-determined, it has essentially different mathematical properties from a Laplacian, in spite of the superficial similarities. For example, with ODETLAP, unlike with a Laplacian, interior local extrema can be inferred. Fitting a nested set of contours produces a rounded hill, not a flat topped mesa. Also, unlike with a Laplacian PDE, the inferred surface is more continuous across lines of known points, and so, the data points are not as visible (or not at all visible) in the generated surface, as shown in Figure 1.1.

The least squares solution to this system can be biased by weighting the different equation classes differently by changing Equation 1.2 above to

$$4Rz_{ij} = R\left(z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i.j+1}\right) \tag{1.4}$$

The multiplicative parameter R trades off smoothness (large R) versus accuracy (small R) at that point. This concept also allows inconsistent datasets of varying accuracies (and hence weights) to be conflated.

In the second component of ODETLAP, compressing an array of elevations, we select some of the points, perhaps 1%, as *known* points, and ignore the others. In some sense, the known points are more important. They might be mountain tops, valley bottoms, the ends of ridges, river confluences, etc. That set of known points, $\mathscr{S} = \{(x_i, y_i, s_i)\}$, is the compressed representation of the data. Applying the first component of ODETLAP reconstructs an approximation to the original data.

There is no Equation 1.4 for points on the border of the region. However those points' values are part of the equations of their nonborder neighbors. One might, wrongly, create an equation for a border point to set it equal to the average of the 2 or 3 neighbors that it does have. However that would impose an unwarranted bias towards horizontality on the surface at the border. To prevent the whole system from being underdetermined, there must be at least as many known points as border points. In practice, this is not a problem.

2D-ODETLAP is used to compress a surface, expressed as an array of elevations, as follows.

1. Select a random subset $\mathscr{S}$ of the points in the input array. Selecting 0.1% of the points is reasonable.
2. Use ODETLAP to compute an initial approximation of the dataset.
3. Of all the points in the array, find those that are farthest from that approximation. Again, 0.1% of the array is a reasonable heuristic.
4. If the (maximum, average, or whatever desired metric) error is adequate, then exit this loop. Otherwise:
5. Insert those worst points into $\mathscr{S}$.
6. Go back to step 2.

The extension of ODETLAP from 2D to 3D or 4D is obvious. Each nonborder point now has 8 or 16, rather than 4, neighbors. In our current 5D-ODETLAP im-

plementation, the fifth dimension in our dataset is a property, of a different nature than the other four, we currently implement a 4D-ODETLAP for each property (e.g., temperature, salinity, ...) in the fifth dimension. Meanwhile the points' locations are constrained to be the same in each 4D-ODETLAP. This utilizes the fact that the locations of important points for one property are often also important points for the other properties. Therefore we need to encode the points' locations only once, saving considerable space. The representation of the compressed dataset is the set of positions and vector of values of the elements of $\mathscr{S}$.

We encode the known point locations and the respective property values separately. Note that the bitmap of the positions is like a 2-value facsimile image. For both the set and vector, we tested various methods to optimize the coding. We used LEDA, a C++ class library for efficient data types and algorithms [16], to build our own entropy coder. LEDA contains a wide range of simple coders, which could be easily combined; we built the following ten different coders.

- 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

1.4 Known-Point Position Compression

For the sparse binary 4D matrix representing the positions of the known points, we first used *Binary RLE* (binary run length encoding) to convert it into a integer vector, defined thus:

$$P = \{r_1 r_2 \dots r_i \dots r_k\} \tag{1.5}$$

where r_i represents the number of 0's before the i-th 1 in a row-major order. k is the total number of known points, i.e., the number of 1's.

Figure 1.3 shows the result of using the 17 different coders mentioned in the previous section to compress the 4D binary position file. The 10 coders from LEDA compress the integer vector from *Binary RLE*. We also include the result of the Bzip2 and JPEG 2000 coder for the integer vector. There are five possible coders in TIFF, which is widely used for compressing binary images: the International Telegraph and Telephone Consultative Committee (CCITT) Fax3, CCITTFax4, Lempel-Ziv-Welch

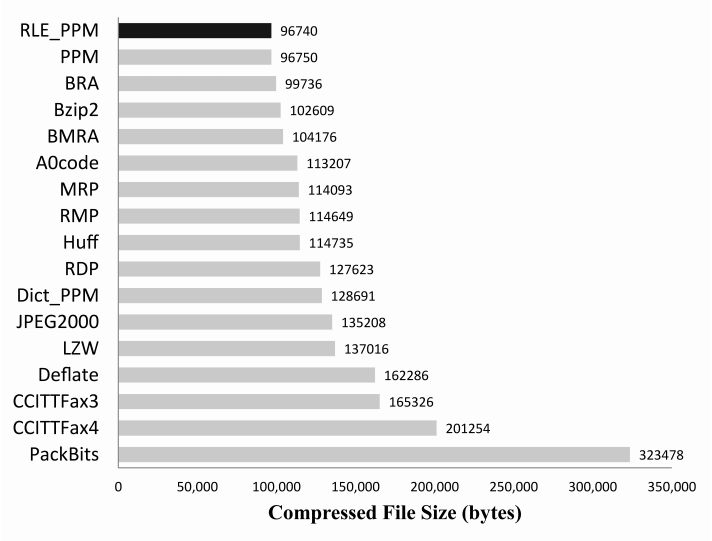

Fig. 1.3 Binary position file compressed file size by 17 different coders. The uncompressed 5D binary position file is 2332800 bytes.

(LZW), PackBits and Deflate. We present the TIFF compression results using them on a test dataset that is a $90 \times 180 \times 24 \times 12 \times 4$ array from WOA 2009, with 91119 known points. The RLE_PPM coder is the winner, and so 5D-ODETLAP uses it.

The known-point position compression is lossless since otherwise the points' positions would move in an unpredictable way. Allowing a lossy compression here is a possible future research idea.

1.5 Known-Point Value Compression

For the vector of values for the known points, we used Lloyd relaxation to produce an optimal quantization for the floating point value. The best-known and earliest quantization algorithm, generalized Lloyd algorithm (GLA) [13] is based on minimizing the Mean Square Quantization error (MSE_q), which can be expressed as:

$$MSE_q = \sum_{i=1}^{N} \int_{d_i}^{d_{i+1}} (x - C(x))^2 f_x(x) dx \tag{1.6}$$

where x and $C(x)$ are input floating value and 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 file contains the integer indices, while the other records floating point codebook.

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

Table 1.1 Influence of delta encoding on the size of the final compressed floating point value vector. The test data is the same as in Figure 1.3.

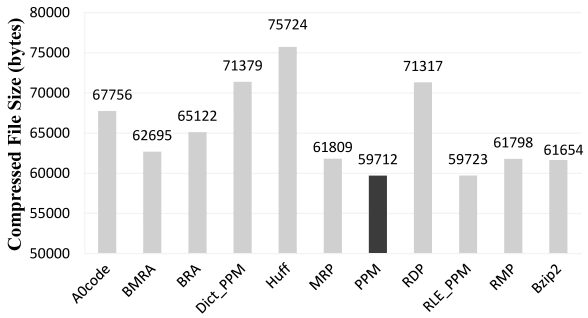

Fig. 1.4 Floating point value vector final compressed file size by 11 different coders.

After the quantization step, we apply a conditional delta encoding to the quantized integer indices. This part of the encoding is conditional because we apply it only when it benefits the compression. In particular, this step adds one sign bit to the integer vector. As shown in Table 1.1, this action usually improves the coding efficiency of the subsequent PPM coding only when adding such a bit allows us to store a value whose number of bits is a multiple of 4 or 8 (because the PPM coder acts on a byte stream).

Figure 1.4 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 1.3. We also include the result of the Bzip2 coder for comparison. This figure demonstrates that the PPM coder performs the best in our 5D-ODETLAP compression framework.

1.6 Challenge of Compute-Intensiveness

The biggest challenge of this compression framework is that solving overdetermined sparse linear systems is compute-intensive. The normal equations transformation,

taking $Ax = b$ to $A^T Ax = A^T b$, reduces the computation time considerably. Indeed the system is now exactly determined, instead of overdetermined. However, this increases the memory requirement, and the computation is still slow. For example, solving a 104976×104976 linear system takes up to 58 GB main memory and about 1.8 hours on a 2006-vintage workstation with four 2.4GHz processors and 60 GB of main memory running Ubuntu 10.04.2 and 64-bit MATLAB R2009a. Current processors might improve that by a small integer factor. The following sections show how modern GPU accelerator technologies give a large factor improvement.

A major sub-challenge is integrating various software tools described below into an efficient solution. Not having to reinvent those tools allows us to concentrate on our problem.

Those tools reduce the computation time so much that now the I/O time for transmitting data to the graphics processor is the dominant cost. So, the major future challenge will be to reduce that.

1.7 Application of Accelerator Technology

Modern accelerator technology is significant in this research because of the greatly increased potential performance for this CPU-bound application. Fortunately, our application is already amenable to parallel solution.

1.7.1 CUDA-Based Solver Introduction

The Compute Unified Device Architecture (CUDA) [19] by NVIDIA provides a widely used developer-friendly General Purpose Computing on Graphics Processing Units (GPGPU) interface. CUSP [2] is an open source library for sparse linear algebra computations using CUDA. It provides a flexible, high-level interface for manipulating sparse matrices and solving sparse linear systems. The CUSP library contains two iterative linear solvers, the Conjugate Gradient solver (CG) and Biconjugate Gradient Stabilized solver (BICGS) with an optional Jacobi preconditioner. With a proper construction of the linear system, a simple function call in MATLAB can solve the linear system using the GPU computing power without any prior knowledge about CUDA GPU programming. Indeed, that is one of the messages of this paper: that these tools exist and are useful. Figure 1.5 shows an outline of the solver.

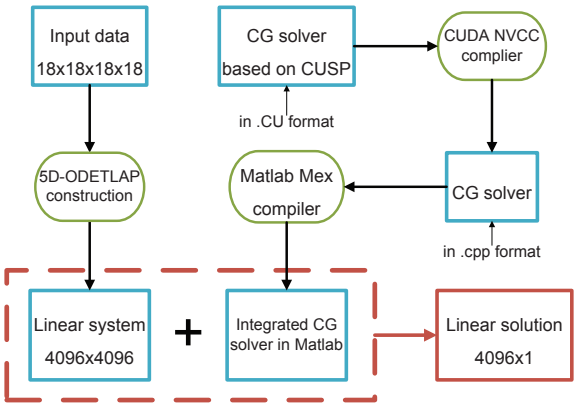

Fig. 1.5 Outline of the Solver

1.7.2 Solver Selection and MATLAB Integration

MATLAB's Cholesky Factorization (CF) exact solver is compute-intensive. However a precise solution is not required since 5D-ODETLAP is lossy. The conjugate gradient (CG) iterative solver in the CUSP library may work better.

First, we tested the CUSP solver on a linear system of size 234256×234256 constructed by 5D-ODETLAP. The CUSP CG solver, taking 179 seconds, was much more efficient than the CF direct solver (49237 seconds) and CG solver (5495 seconds). However, of that 179 seconds, only 9 seconds was spent on actually solving the linear system. The remaining 170 seconds was spent on the data transfer between 5D-ODETLAP, implemented in MATLAB, and the CUSP solver, a C++ executable. To improve this situation, we utilized the MATLAB Executable interface (MEX), which allows users to interface C, C++ or Fortran subroutines to MATLAB. MEX-files are a way to call the custom C, C++ or FORTRAN routines directly from MATLAB as if they were MATLAB built-in functions. Therefore we can largely reduce the overhead for transferring data between the MATLAB program and the CUSP library. But since the CUSP library is implemented both in C++ and CUDA, this adds a certain amount of complexity to incorporate it into the MEX file. Fortunately, CUDA compiler allows users to compile CUDA code into C++ code as an intermediate step using *nvcc –cuda*. So first we write the CUSP code in a MEX style. Then this mixed source code will be compiled into C++ code, which can then be compiled into a MEX file and called directly from the MATLAB program.

In Table 1.2, a comparison of solvers in MATLAB and CUSP demonstrates that for the linear system from 5D-ODETLAP, the CUSP CG solver with Jacobi preconditioner runs the fastest. The linear systems in this table are constructed from original 4D datasets ranging from 8^4 to 18^4, so the resulting linear systems' size ranges from 4096×4096 to 104976×104976. This solver runs more than seven times faster (39.67:4.84) than its CPU counterpart with the same residual size of

Table 1.2 Effective solver time efficiency comparisons between the MATLAB Cholesky Factorization (CF) direct solver, MATLAB CG solver, CUSP CG solver, CUSP CG solver with Jacobi preconditioner, CUSP BICGS solver with preconditioner and CUSP CG solver. The time measurement is in seconds.

System Size	Cholesky Factor. Solver	MATLAB CG	CUSP CG	CUSP CG Jacobi	CUSP BICGS Jacobi
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

their solutions in the test linear system of size 104976×104976. Also, it is more than 1340 times faster than the CF direct solve in MATLAB. 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 device memory on GPU on the workstation described above.

1.8 Comparison with JPEG 2000 and 3D-SPIHT

We used the two real world geospatial 5D datasets mentioned above in Section 1.1 to test our 5D-ODETLAP framework. We geographically divided these two datasets into 8 different datasets each of size $90 \times 180 \times 24 \times 12 \times 4$, named WOA05_1, WOA05_2, WOA05_3, WOA05_4, WOA09_1, WOA09_2, WOA09_3 and WOA09_4. This allowed us to test the robustness of 5D-ODETLAP on 8 distinct datasets, to see how sensitive it is to the particular data. The property values were pre-truncated to single precision floats.

Since 3D-SPIHT and JPEG 2000 are among the most popular lossy compression methods, we compared 5D-ODETLAP with them. For a fair comparison, we used binary search in 3D-SPIHT to find a suitable bit rate to produce results with the same mean percentage error in the one test and the same maximum percentage error in the other. 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 applied 3D-SPIHT on each of these 3D datasets and measured the error together as a 5D reconstruction in decompression.

Similarly, we used binary search to find a compression ratio in JPEG 2000 to obtain results with the same error in both cases. We also applied JPEG 2000 to each 90×180 2D dataset of the overall $24 \times 12 \times 4$ number of 2D datasets. In addition,

Table 1.3 Compression comparison between 5D-ODETLAP, JPEG 2000 and 3D-SPIHT with the same **mean** percentage error on eight different datasets. The last two columns show the ratio of 5D-ODETLAP's compression ratio over the compression ratio of JPEG 2000 and 3D-SPIHT respectively.

Dataset	% Fixed Mean Err	% Max Err, JPEG 2000	% Max Err, 3D-SPIHT	% Max Err, 5D-ODETLAP	Ratio $(\frac{5D-ODETLAP}{JPEG2000})$	Ratio $(\frac{5D-ODETLAP}{3D-SPIHT})$
woa05_1	1.42	44.83	66.46	10.41	8.16	2.40
woa05_2	1.48	49.33	59.18	9.35	9.56	3.61
woa05_3	1.47	65.56	80.23	8.94	4.24	1.13
woa05_4	1.56	67.56	74.14	10.81	8.57	2.41
woa09_1	1.46	48.13	68.18	9.02	8.18	2.41
woa09_2	1.49	51.21	62.15	8.77	9.80	3.75
woa09_3	1.54	75.00	79.35	11.13	4.24	1.14
woa09_4	1.58	65.55	71.50	11.58	8.58	2.42

Table 1.4 Compression comparison between 5D-ODETLAP, JPEG 2000 and 3D-SPIHT with approximately the same **maximum** percentage error on eight different datasets. The last two columns show 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, JPEG 2000	% Mean Err, 3D-SPIHT	% Mean Err, 5D-ODETLAP	Ratio $(\frac{5D-ODETLAP}{JPEG2000})$	Ratio $(\frac{5D-ODETLAP}{3D-SPIHT})$
woa05_1	10.41	0.52	0.37	1.42	16.33	11.51
woa05_2	9.35	0.32	0.35	1.48	24.02	15.74
woa05_3	8.94	0.26	0.28	1.47	13.84	9.47
woa05_4	10.81	0.37	0.34	1.56	22.23	14.91
woa09_1	9.02	0.39	0.29	1.46	18.97	13.87
woa09_2	8.77	0.31	0.36	1.49	24.43	15.51
woa09_3	11.13	0.36	0.31	1.54	12.05	8.91
woa09_4	11.58	0.39	0.34	1.58	21.92	15.22

since JPEG 2000 takes only unsigned 1, 8 or 16 bit integer input, we first used uniform quantization to reduce each input 2D dataset to 16 bit unsigned integer. Thus, all the three methods had the same mean or maximum percentage error on all the 8 test datasets, again for a fair comparison.

Table 1.3, with data from [11], compares the three methods for the same *mean* percentage error. The first observation is that the maximum percentage error for 5D-ODETLAP is much smaller than that of both JPEG 2000 and 3D-SPIHT. This advantage of 5D-ODETLAP is credited to its iterative greedy sampling process, since it eliminates the points with the largest error at each iteration by adding them into the known-point set. The JPEG 2000 and 3D-SPIHT methods do not have this adaptability, and thus produce a much larger maximum percentage error. Second, the compression ratio of 5D-ODETLAP 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.

Table 1.4, with data from [11], shows the result of forcing the *maximum* percentage error to be the same for all three methods. The result may be useful in scenarios in which, for example, users need to have a compressed file with guaranteed no more than 10% error. 5D-ODETLAP's mean error is larger than the others because it spreads out the error more evenly. However 5D-ODETLAP's compression ratio is even larger than in the previous fixed mean percentage error case. Here, the compression ratio of JPEG 2000 and 3D-SPIHT will be considerably worse than 5D-ODETLAP, since the mean error of the compressed file is unnecessarily small. Comparatively, 5D-ODETLAP's iterative sampling process ensures that the points with largest error are always selected, which reduces the maximum error at each step.

1.9 Conclusion and Future Research Plan

5D-ODETLAP demonstrates the efficient application of a massively parallel GPU to an important GIS problem — compressing multidimensional GIS data. 5D-ODETLAP also exploits spatial and temporal redundancies in the data better than previous methods. 5D-ODETLAP's potential impact extends to multidimensional datasets in other domains, such as computational fluid dynamics (CFD). The NVIDIA GeForce 9800GT, the GPU used in this paper, is several years old; we anticipate even better results with a current graphics processor.

Currently 5D-ODETLAP only partially exploits correlations between 4D data layers within a 5D dataset. Also, its coding of the bitmap denoting the points in $\mathscr{S}$, and its compression of the floating values at those points might be improvable. Finally, overdetermined extensions to other PDEs remain to be investigated. Therefore, we expect even better compression of high-dimensional data in the future.

1.10 Acknowledgement

This research was partially supported by NSF grants CMMI-0835762 and IIS-1117277.

References

1. Anagnostou, K., Atherton, T.J., Waterfall, A.E.: 4d volume rendering with the shear warp factorisation. In: Proceedings of the 2000 IEEE symposium on Volume Visualization, VVS '00, pp. 129–137. ACM, New York, NY, USA (2000). DOI http://doi.acm.org/10.1145/353888.353909
2. Bell, N., Garland, M.: CUSP: Generic Parallel Algorithms for Sparse Matrix and Graph Computations. http://cusp-library.googlecode.com (2010). Version 0.1.0
3. Bjøke, J.T., Nilsen, S.: Efficient representation of digital terrain models: compression and spatial decorrelation techniques. Computers & Geosciences **28**(4), 433–445 (2002). DOI DOI: 10.1016/S0098-3004(01)00082-6
4. Franklin, W.R.: The RPI GeoStar project. In: 25th International Cartographic Conference. Paris (2011)
5. Franklin, W.R., Inanc, M., Xie, Z.: Two novel surface representation techniques. In: Autocarto 2006. Cartography and Geographic Information Society, Vancouver Washington (2006)
6. Franklin, W.R., Said, A.: Lossy compression of elevation data. In: Seventh International Symposium on Spatial Data Handling. Delft (1996)
7. Inanc, M.: Compressing terrain elevation datasets. Ph.D. thesis, Rensselaer Polytechnic Institute (2008)
8. Kidner, D.B., Smith, D.H.: Advances in the data compression of digital elevation models. Computers & Geosciences **29**(8), 985–1002 (2003). DOI DOI: 10.1016/S0098-3004(03)00097-9
9. Kim, B.J., Pearlman, W.: An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPIHT). In: Data Compression Conference, 1997. DCC '97. Proceedings, pp. 251–260 (1997). DOI 10.1109/DCC.1997.582048
10. Lalgudi, H., Bilgin, A., Marcellin, M., Nadar, M.: Compression of fMRI and ultrasound images using 4D SPIHT. In: Image Processing, 2005. ICIP 2005. IEEE International Conference on, vol. 2, pp. II – 746–9 (2005). DOI 10.1109/ICIP.2005.1530163
11. Li, Y.: CUDA-accelerated HD-ODETLAP: a high dimensional geospatial data compression framework. Ph.D. thesis, Rensselaer Polytechnic Institute (2011)
12. Li, Y., Lau, T.Y., Stuetzle, C., Fox, P., Franklin, W.R.: 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 (2010). (PhD Dissertation showcase)
13. Lloyd, S.: Least squares quantization in PCM. Information Theory, IEEE Transactions on **28**(2), 129–137 (1982). DOI 10.1109/TIT.1982.1056489
14. Locarnini, R.A., Mishonov, A.V., Antonov, J.I., Boyer, T.P., Garcia, H.E., Baranova, O.K., Zweng, M.M., Johnson, D.R.: World ocean atlas 2009, volume 1: Temperature p. 184 (2010)
15. Lum, E.B., Ma, K.L., Clyne, J.: Texture hardware assisted rendering of time-varying volume data. In: VIS '01: Proceedings of the conference on Visualization '01, pp. 263–270. IEEE Computer Society, Washington, DC, USA (2001)

16. Mehlhorn, K., Näher, S.: LEDA: a platform for combinatorial and geometric computing. Commun. ACM **38**(1), 96–102 (1995). http://www.mpi-sb.mpg.de/guide/staff/uhrig/leda.html
17. Menegaz, G., Thiran, J.P.: Lossy to lossless object-based coding of 3-d mri data. IEEE Transactions on Image Processing **11**(9), 1053–1061 (2002). DOI 10.1109/TIP.2002.802525
18. Muckell, J.: Evaluating and compressing hydrology on simplified terrain. Master's thesis, Rensselaer Polytechnic Institute (2008)
19. NVIDIA: NVIDIA Corporation: Compute Unified Device Architecture Programming Guide. http://developer.nvidia.com/cuda (retrieved 1/11/2011)
20. Plaza, A., Plaza, J., Paz, A.: Improving the scalability of hyperspectral imaging applications on heterogeneous platforms using adaptive run-time data compression. Computers & Geosciences **36**(10), 1283–1291 (2010). DOI DOI: 10.1016/j.cageo.2010.02.009
21. Sanchez, V., Nasiopoulos, P., Abugharbieh, R.: Lossless Compression of 4D Medical Images using H.264/AVC. In: 2006 IEEE International Conference on Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings, vol. 2, p. II (2006). DOI 10.1109/ICASSP.2006.1660543
22. Stookey, J.: Parallel terrain compression and reconstruction. Master's thesis, Rensselaer Polytechnic Institute (2008)
23. Stookey, J., Xie, Z., Cutler, B., Franklin, W.R., Tracy, D.M., Andrade, M.V.: Parallel ODETLAP for terrain compression and reconstruction. In: W.G. Aref, et al. (eds.) 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS 2008). Irvine CA (2008)
24. Taubman, D.S., Marcellin, M.W., Rabbani, M.: Jpeg2000: Image compression fundamentals, standards and practice. Journal of Electronic Imaging **11**, 286 (2002). DOI doi:10.1117/1.1469618
25. Tracy, D.M.: Path planning and slope representation on compressed terrain. Ph.D. thesis, Rensselaer Polytechnic Institute (2009)
26. Xie, Z.: Representation, compression and progressive transmission of digital terrain data using over-determined laplacian partial differential equations. Master's thesis, Rensselaer Polytechnic Institute (2008)
27. Yang, W., Lu, Y., Wu, F., Cai, J., Ngan, K., Li, S.: 4-D wavelet-based multiview video coding. IEEE Transactions on Circuits and Systems for Video Technology **16**(11), 1385–1396 (2006)
28. Ziegler, G., Lensch, H., Magnor, M., Seidel, H.P.: Multi-video compression in texture space using 4d spiht. In: Multimedia Signal Processing, 2004 IEEE 6th Workshop on, pp. 39 – 42 (2004). DOI 10.1109/MMSP.2004.1436410