An Adaptive Sub-Sampling Method for in-memory Compression of Scientific Data

Didem Unat, Theodore Hromadka III, Scott B. Baden
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, USA
{*dunat,thromadk,baden*}@*cs.ucsd.edu*

Abstract

A current challenge in scientific computing is how to curb the growth of simulation datasets without losing valuable information. While wavelet based methods are popular, they require that data be decompressed before it can analyzed, for example, when identifying time-dependent structures in turbulent flows. We present Adaptive Coarsening, an adaptive subsampling compression strategy that enables the compressed data product to be directly manipulated in memory without requiring costly decompression. We demonstrate compression factors of up to 8 in turbulent flow simulations in three dimensions. Our compression strategy produces a non-progressive multiresolution representation, subdividing the dataset into fixed sized regions and compressing each region independently.

1. Introduction

Long term improvements in system performance have enabled significant advances in our ability to employ numerical models to understand physical phenomena. However, these developments have come at the cost of a flood of data that is becoming increasingly difficult to analyze, and users are compelled to compress the data using methods that are often ad-hoc.

Due to the presence of noise in floating point data, lossless compression [1], [2] offers only modest improvement, and lossy compression is employed in practice. The majority of the published compression techniques involve data visualization [3], [4] and include multiresolution techniques such as wavelets [5], [6].

We are interested in compressing data that will be subsequently analyzed by a process that includes both segmentation and numerical differentiation. The tolerances in taking numerical derivatives are much higher than those in viewing visual media. The analysis algorithms are time consuming, and it is important to realize compression both in memory as well as on disk. Wavelet compression offers help with the latter goal, but not the former. Data must be de-compressed prior to segmentation, and other operations (such as taking spatial derivatives) will have to be intrusively recoded to operate in the wavelet basis set.

We present Adaptive Coarsening (AC), a compression technique that captures data selectively and locally. It builds on Belfor and Hesps' adaptive subsampling method for High Definition Television [7], with extensions appropriate to studying flows that have been modeled with partial differential equations. Like wavelet compression [6], AC is a multi-resolution technique; however,

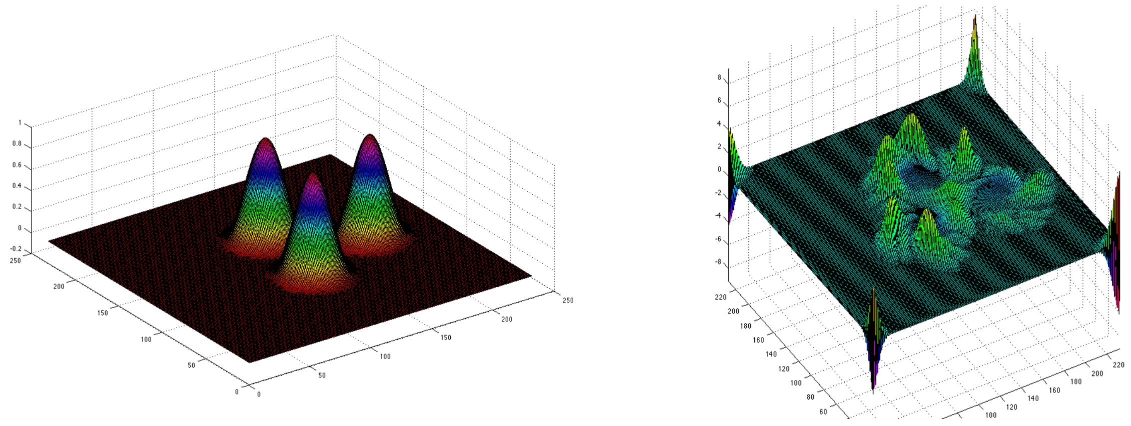

Figure 1: Solution to the Navier Stokes equations in two dimensions. The figures plot vorticity, which is the curl of velocity. The flow field becomes more irregular over time, resulting in a loss in compression.

it does not represent data progressively. As a result, the compact data products can be numerically differentiated and subsequently segmented, without first having to be de-compressed. Compared with wavelet-based techniques, the advantages of our approach are considerable savings in space and time costs for post-processing and the re-use of existing analysis code.

The representation used in AC is reminiscent of that employed in solving partial differential equations with structured adaptive mesh refinement (SAMR) methods [8], [9]. SAMR represents data as a patchwork of meshes at varying resolutions, according to the local smoothness properties of the data. AC employs similar notions, estimating errors committed in downsampling the data.

We present results for datasets coming from solutions to the Navier-Stokes Equations in two and three dimensions. We obtained compression factors of up to 25 and 8, respectively, the lower compression factor for a turbulent flow study. In that study, the data exhibit a wide range of length scales and hence are difficult to compress.

Our paper makes three contributions. The first is an *in-memory* compression technique that, unlike wavelet methods, avoids the need to decompress the data prior to manipulation in memory. The second is an adaptive error estimation procedure appropriate for solutions to Partial Differential Equations, that takes into account anisotropic behavior in the solution, and is not subject to real time constraints (as in HDTV). The third is a set of heuristics needed to subsample fluid flows effectively, including the local number of degrees of freedom present in the compressed data.

Our compressed data products include geometric meta data describing the structure of a multi-resolution mesh, which may be used to optimize subsequent data access and analysis. AC has a modest cost, and it parallelizes trivially.

2. Adaptive coarsening

Consider the numerical solution to a time-dependent partial differential equation in two dimensions, shown in Fig.2, which has been computed on a uniform mesh using a finite difference method.

Some regions of the solution are smoother than others, and we can use knowledge about local smoothness to compress the data. Fig.2 depicts a mesh rendered into patches of uniform physical extent; each patch carries a constant mesh resolution which may differ from others'. Some patches will need to be stored at full resolution, while others can be captured with fewer degrees of freedom. It is also possible to employ variable size patches (in 1D [10], [11]) or to dispense with patches entirely [12], [13], in effect allowing the patches to degenerate to a single point. Whatever approach we choose to render the patches, the algorithm for determining them is the same: we subsample an area of interest to the maximum extent that the original underlying data can be reconstructed to the specified error tolerance.

Adaptive subsampling employs the following error estimation procedure to determine which points may be sampled at a lower resolution. Let G^h be a set of points spaced evenly over a discrete index space which is a subset of Z^n. In two dimensions we have ordered pairs of integers. Let $u(G^h)$ or, by an abuse of notation, u^h, represent the data on G^h. Define the *subsampling* operator $\mathcal{R} : u^h \rightarrow u^{2h}$ which maps values from a mesh defined on G^h onto a sub-sampled index domain G^{2h}, which has one-half as many points in each dimension. We define the *upsampling* operator $\mathcal{P} : u^{2h} \rightarrow u^h$, mapping values from a subsampled (coarser) index domain G^{2h} onto the refined domain G^h. Both $\mathcal{P}$ and $\mathcal{R}$ may be customized to the data and the upsampling factor may be greater than two.

The algorithm proceeds as follows. We compress the data and then re-construct the original by upsampling, i.e. interpolation. That is, we compute $\mathcal{P} \circ \mathcal{R}\ (u^h)$. Noting that $\mathcal{P} \circ \mathcal{R}$ is not equivalent to the identity operator, we define the *error* operator $\mathcal{E} = \mathcal{I} - \mathcal{P} \circ \mathcal{R}$. Now, for a given relative error threshold ϵ, we subsample the mesh where $|\mathcal{E}(u^h)/u^h| < \epsilon$ (relative error), where $|\cdot|$ is the absolute value taken point-wise. We carry out this procedure recursively on the newly-subsampled part of the mesh until it is no longer possible to subsample the data without violating the accuracy threshold when we reconstruct the data at finest resolution, or when the mesh drops below a specified minimum size. Pseudocode for the AC algorithm appears in Fig. 2.

```
1. let level[0] = $u^h$
2. for i := 1 to maxLev
3.     let $h^i$ := discretization at level $i$
4.     foreach subdomain $u_j^{h^{i-1}} \in$ level[i-1]
5.         where $(\mathcal{E}(u_j^{h^{i-1}})/u_j^{h^{i-1}}) < \epsilon$,
5.                 create a new coarse domain $u^{h^i}$
6.         level[i] := level[i] $\cup$ $u^{h^i}$
7.         level[i-1] := level[i-1] $\setminus \mathcal{P} u^{h^i}$
8.     end foreach
9. end for
```

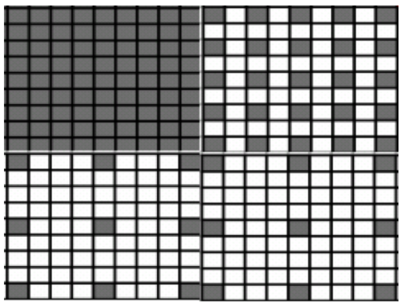

Figure 2: Left: the AC algorithm. Right: AC divides the mesh into uniform patches of constant resolution. Patches need not all have the same resolution.

The algorithm proceeds one level at a time (line 2), starting from the initial (finest) level 0. Line (5) estimates the error and subsamples the mesh accordingly, generating a list of rectangles (solids in 3D, etc.). Since there may be more than one subdomain at the parent's level, we augment the existing subsampled grids at the current level with the union $\cup$ operation (6). Since each point may appear in at most one level we remove any points subsampled from the previous finer level, using the set difference operation $\setminus$ (7).

3. Algorithmic variants

Multi-dimensional datasets pose special challenges that do not arise in a single dimension: they can exhibit different numerical characteristics along different coordinate directions. For example, data may be smoother in some directions than others, and we may be able to improve compression by subsampling at different rates in different directions, that is, by subsampling anisotropically.

To determine the optimal subsampling rate, we employ a greedy algorithm. We enumerate each possible combination of subsampling rate over all axes, and we rank the combinations according to the compression factor, choosing the combination corresponding to the highest compression. For example, if the data changes more rapidly in the x-direction than the y-direction, then we can satisfy the error tolerance with fewer subsampled points in the y-direction than in x. At present we use simple subsampling for $\mathcal{R}$, that is, selecting every Cth point, where C is the subsampling rate.

As mentioned previously, our technique requires an error estimation procedure to choose the appropriate level of compression. Depending on the characteristics of the data and the size of the patches, there are different options for choosing $\mathcal{P}$, the upsampling operator (interpolation), which is used to estimate the error. We currently use linear, polynomial, piecewise cubic spline with natural boundary conditions (the natural boundary condition sets the second derivative to zero at the boundary points), and Akima interpolation.

Not surprisingly, there is also an advantage in applying interpolation schemes anisotropically. In our *mixed* strategy, we may select a different interpolation method along each dimension. The mixed strategy attempts, for each patch, all possible specified interpolation methods along each direction and it chooses the combination that results in the highest compression factor. The mixed strategy can improve significantly on the uniform strategy of picking a single interpolation method. Consider a 3D 17^3 patch with sampling rates of 16, 4, and 4 in the X, Y, and Z directions, respectively. The subsampled patch has 2 points in the X-direction, 5 points in Y and Z. Since higher order interpolation procedures may require 5 points, the small extent of the patch in the X-direction rules out higher order interpolation along that direction. We are forced to use a lower order interpolation procedure such as linear interpolation. Even though there is an accompanying loss of accuracy along the X-direction, and hence some loss in compression, we are able to employ higher order interpolation along the other directions, and do better than if we used only linear interpolation along all axes.

4. Results

We present results for 2D and 3D datasets obtained by solving time-dependent partial differential equations. We compare the benefits of adaptive coarsening against wavelet compression. Both datasets comprise a series of snapshots in time. Due to the onset of turbulence, the data become more irregular with time, with a concomitant drop in compression, as shown in Fig. 4.

The 2D data set was obtained by solving the Navier Stokes equations.[1] There were 51 snapshots, each comprising an array of 1020×1020 floating point numbers. These data represent a differentiated quantity–vorticity–which is obtained by taking the curl of velocity. It is a scalar in two dimensional flow. The 3D data were came from a study of mixing of material in stratified flows with multiscale dynamics involving coupled evolution of coherent structures, turbulence and internal waves [14]. The data set contains 11 snapshots of dimension $641 \times 194 \times 386$ (1.14×10^9 bytes) containing velocity data. We also produce vorticity, which is a 3D vector quantity in 3D flows.

Because there is a tradeoff of accuracy and compression, we calibrate our results for "engineering precision," which is sufficient for analyzing flow features. In particular, we choose accuracy thresholds $\epsilon = 10^{-2}$ and 10^{-3}, that is, we require that we be able to reconstruct all points of the mesh to within the relative error ϵ, that is, in the L_∞ norm. (By comparison,

1. `http://www.math.ucla.edu/~anderson/270e.1.04f/Assignment8/Assign8.html.`

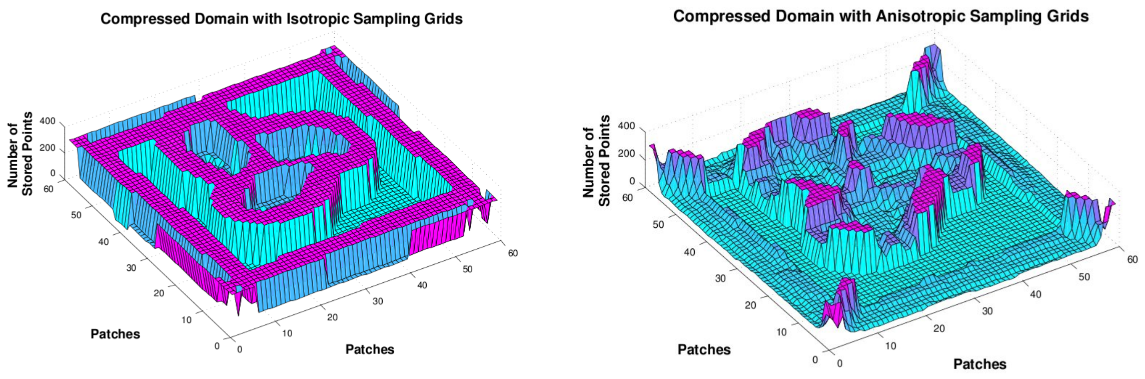

Figure 3: Anisotropic subsampling improves compression. The number of stored points in a 2D dataset compressed with isotropic sampling (left) and anisotropic sampling (right). The height of the bars is proportional to the number of retained points; thus, lower bars indicate a higher degree of compression.

video compression can tolerate large pointwise errors. Thus, an averaged quantity is appropriate, resulting in increased compression.) Since we use the relative error as the error metric, we need to protect against division by zero or other small numbers. We specify a *cutoff* value; when values are smaller than cutoff, we take the absolute error rather than the relative error. This cutoff value is used to distinguish very small values from noise and may be dataset-dependent. We determined that 10^{-5} is a good cutoff value in our datasets, which is consistent with suggestions made by the providers of the data.

We will first demonstrate the benefits of anisotropic and mixed interpolation. For the 3D data, we look at just one component of velocity (x). Later, when we compare AC against wavelet compression, will look at both vorticity and velocity, considering all 3 vector components.

Anisotropic subsampling. By employing anisotropic subsampling in two dimensions, we were able to boost compression by about a factor of 2, as show in Tab. 4. This improvement is further illustrated in Fig. 4(a) and (b), which maps the number of points retained in each patch over a 2D domain. In particular, some patches require higher resolution than others and some cannot be compressed. This phenomenon occurs where the solution is irregular. Such irregularity arises near the physical boundaries and in the vicinity of the 3 peaks shown on the right side of Fig. 2. In order to maintain the efficiency of our search strategy we sorted the subsampling schedule in increasing order of the aggregate subsampling rate along all directions. We start from the level giving lowest resolution and stop when we reach a level that satisfies the error tolerance.

Mixed Interpolation. Tab. 4 compares the effectiveness of employing different interpolation schemes. “Linear” corresponds to pure bilinear interpolation, “Spline” is piecewise cubic spline with natural boundary conditions. The “Mixed” scheme selects dimension and patch specific interpolation as described previously. Mixed interpolation provides a smaller improvement in 3D than in 2D, but by using it together with anisotropic sampling, we obtained compression factors of 25 in 2D and 7.4 in 3D, at a tolerance of $\epsilon = 10^{-3}$. By comparison, these factors dropped to 2.3 and 3.34, respectively, when isotropic sampling and spline interpolation were used.

AC chooses the sampling rate and interpolation independently for each dimension. To speed up the search time for the optimal strategy, we use an heuristic based on the observation that neighboring patches are highly likely to use the same interpolation scheme and sampling rate. Therefore, we evaluate the compression rate and scheme suggested by the previously compressed patch and apply our greedy algorithm if the suggestion does not help.

Table 1: Compression factors with anisotropic and isotropic sampling taken over the full simulation. The ratio is the improvement obtained with anisotropic sampling.

Datasets	Interpolation Scheme	Anisotropic (Aniso)	Isotropic (Iso)	Ratio: Aniso/Iso
	Linear	3.07	1.67	1.85
2D	Spline	4.60	2.26	2.04
(10^{-3})	Mixed	**25.1**	9.06	**2.77**
	Linear	11.0	7.00	1.57
3D	Spline	13.2	8.71	1.51
(10^{-2})	Mixed	**13.9**	9.05	**1.53**
	Linear	5.03	2.74	1.84
3D	Spline	6.58	3.44	1.91
(10^{-3})	Mixed	**7.44**	4.35	**1.71**

Factors that affect compression. The achieved compression factor in AC depends on the patch size, cutoff value, the tolerance, and the dataset. We observed that the patch size should be small enough to adapt to the local spatial frequency but large enough to enable the use of higher order interpolation algorithms. We used subsampling rates that are integer powers of 2. Thus, patch sizes that are odd numbers of the form 2^n+1 are appropriate. Fig. 4 shows the effect of different patch sizes on the 3D dataset.

Compression of vector quantities. The results reported in Tab. 4 establish the benefits of anisotropic compression together with mixed mode interpolation, but report just a single component of velocity and vorticity (vorticity is a scalar in two dimensions, and a 3D vector in three). We next report the overall compression obtained when all three components of velocity and vorticity are compressed in the 3D dataset. This gives a truer measure, since the solution is smoother in the X direction than in the Y and Z directions. Moreover, we use both velocity and vorticity in computing derived quantities for flow analysis. Because vorticity is obtained by differentiating velocity, we expect–and observe–lower compression factors than for velocity. We compare against the wavelet compression, which is a popular technique. (Details described below.)

Tab. 4 displays compression by component as well as an overall quantity when all 3 components were taken together. Overall, we were able to compress velocity and vorticity by a factor of 8.2 and 5.1 respectively, when $\epsilon = 10^{-2}$. By comparison, the wavelet method realized compression factors of 5.6 and 6.5. At this tolerance, AC is highly competitive with wavelet compression, since it realizes compression in memory as well as on disk. Wavelet compression exhibited the same trends as did AC; compression in the X component exceeded that in Y and Z, and the data compressed less and less as turbulent structures became more and more irregular.

When we increase ϵ to 10^{-3}, AC compression drops to 3.4 and 3.0, respectively. Wavelet compression provides approximately two times the compression on disk; however, AC enjoys the advantage of a three-fold compression in memory, which is significant.

Wavelet Compression. We used a variety of wavelet compression techniques in the *MATLAB r2007a Wavelet Toolbox*. We explored a representative set of wavelet functions, and found that the *Haar* function performed the best under our test conditions. We compressed the 3D data a plane at a time; we decomposed the data with *wavedec2,* setting α to the compressive standard 1.5 for *Birge-Massart* thresholding using *wdcbm2,* and transformed it with level-dependent *wdencmp2* (α directly affects the error committed in compression). The relative error was measured by comparing against the de-compressed data against the original. The L_∞ (Max) norm of the error

Table 2: Compression for the 3D dataset for vorticity and velocity. The AC compression data show a variation as a function of the tolerance and in the different coordinate directions. "Overall" includes all 3 components. The column labeled u/ω_x presents data for the x component of velocity and vorticity, and so on for y and z. The patch size was 9^3. The tolerance for wavelet compression is a consequence of the choice of the α parameter, as explained below in *Wavelet Compression.*

Method	variable	u/ω_x	v/ω_y	w/ω_z	Overall
AC(10^{-2})	velocity	13.9	6.57	7.05	8.19
	vorticity	6.49	7.30	6.09	5.11
AC(10^{-3})	velocity	7.44	2.54	2.82	3.40
	vorticity	2.56	3.62	3.06	3.02
Wavelet	velocity	6.00	5.46	5.50	5.65
(3.4×10^{-3})	vorticity	6.73	6.39	6.28	6.47

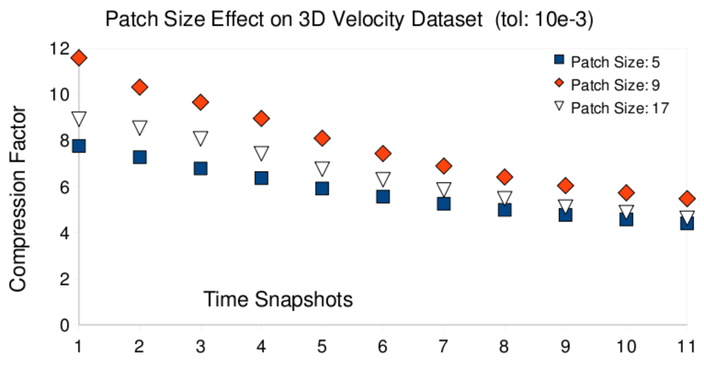

Figure 4: Due to the onset of turbulence, compression diminishes over time in the 3D dataset. The figure also demonstrates the effect of patch sizes on compression.

was 0.017, but dropped to 3.4×10^{-3}, if we exclude the 4 corner points. This revised error is slightly larger than the smaller error threshold used in AC, so we may be overstating the benefits of wavelet compression slightly. Interestingly, we observed ringing effects towards the edges of the simulation volume. Coefficient retention was on average about 85%, so in theory the best compression we could obtain would be 6.6. We used Gnu's *gzip* utility to compress the transformed data and it did reasonably well in compressing the zeroes. A level-dependent encoder may improve compression still further and is under investigation.

Performance. We implemented our compression strategy in C++, and used the following interpolation routines from the Gnu Scientific Library [15]: gsl_interp_linear, $gsl\_interp\_polynomial$, $gsl\_interp\_cspline$ and gsl_interp_akima. We ran on a Beowulf cluster with eight nodes each containing dual AMD 1.8 GHz Opteron 244 CPUs and 2GB of RAM. Processing nodes are interconnected by Gigabit Ethernet. We parallelized the code using MPI [16], and partitioned the data into regions over the processors. Each region consists of a rectangular (possibly square) block of patches. All patches compute independently so there is no communication required while compressing the data. We collected timings with `MPI_Wtime()`.

Our compression library is intended to be employed with a running numerical simulation. Thus, we ignore the costs of loading the data into memory, since that data would normally be memory-resident. Reported running times do *not* include the time to write the output.

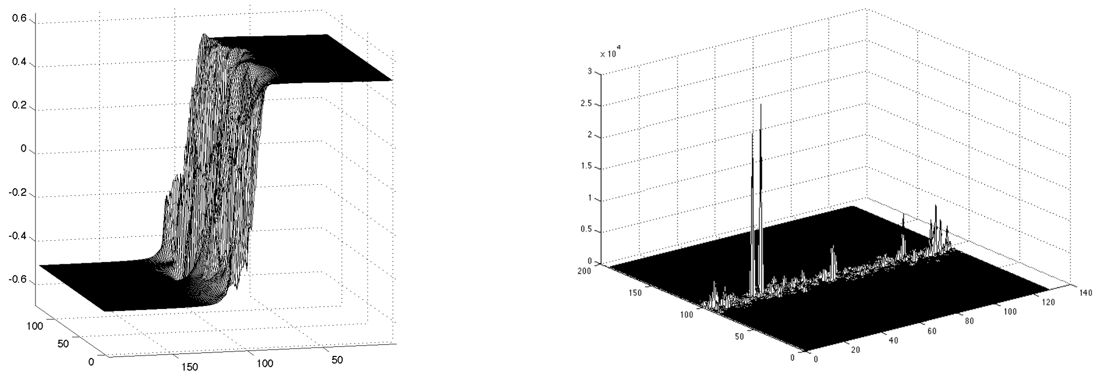

Figure 5: a) Depicting the shear layer and (b) the subsampling error that results from subsampling the entire domain uniformly by a factor of two. The velocity has opposite signs to the left and right of the step. The reported errors are reduced (using the max operator) down the Z axis. The high errors coincide with the sharp change in velocity.

The time to compress each 1.8GB 3D dataset was 5 seconds on 4 Opteron processors. The numerical simulations used to produce the 3D dataset took approximately 120 seconds per *timestep* running on 4 CPUs of an IBM Power 4 system. Although the Opteron cluster is somewhat slower than the Power4 for this workload, the difference is not a factor when assessing overhead costs. In practice, only 1 in 100 (or perhaps 1 in 10) timesteps are written to disk. Thus, the overhead of AC is negligible, a few tenths to a few hundredths of a percent. Because we ran the wavelet compressor in Matlab, direct comparisons of running times are not meaningful.

5. Discussion and conclusions

We have demonstrated a multi-resolution compression technique called Adaptive Coarsening (AC), that enables the user to realize significant *in-memory* compression of numerical data that is to subsequently analyzed. The approach embraces the philosophy that technological factors present an opportunity to employ plentiful processing cycles to reduce the space required to store data.

We are investigating improved sampling techniques. A factor limiting compression is the irregularity in the solution. This especially apparent in the 3D turbulence study dataset. There is shear layer, depicted in Fig. 5, which reduces smoothness significantly in the region surrounding the shear, and bounds overall compression.

Our strategy employs fixed size patches, and builds on an approach proposed for High Definition Television [7]. An alternative is to employ variable sized patches [10], [11], or a fine grained sparse representation that dispenses with the patches entirely that is, with point-wise subsampling [12], [13]. The present strategy results in a more regular structure, and lower software bookkeeping overheads, then either of the other two approaches. However, admitting greater irregularity can also lead to higher compression factors.

Multiresolution compression techniques have been employed for several years in computer graphics, signal processing, and efficient mesh generation [3], [4], [6], [17], [6]. A complete list of references is too lengthly to include here. Adaptive subsampling differs from wavelet-based techniques [5], [6] in that it generates a non-progressive representation and that it enables the direct manipulation of the compressed data sets. Wavelet compression, in contrast, involves many layers of encoding, scanning, and transformations, which prevent direct access to the data without first decompressing it. The decompression process typically requires a reverse of the encoding and transformation steps, which can require loading the entire expanded dataset into memory. There do exist methods that allow partial decompression (e.g. tiling, progressive transforms, and some forms of level-dependent encoding), but these come at some non-trivial cost to the compression ratio, since wavelet compression performs better when it has more global information. We therefore believe that Adaptive Coarsening offers a worthwhile tradeoff in compression ratio, when considering that it permits direct operation on the compressed data without the need for global decompression.

For example, consider segmentation, an important operation in study turbulent flow. Prior to segmentation, we compute various quantities derived from the compressed data. With wavelet compression, we cannot directly segment the wavelet co-efficient, negating the effects of compression in memory. We may be able to reformulate the data analysis algorithms within the wavelet basis, but we can no longer re-use existing high quality analysis software. With adaptive coarsening we retain the benefits of in-memory compression–at a savings in space an in time–as well as existing code bases.

Perhaps the closest relative of AC is the multi-dimensional tree approach of Wilhelms and van Gelder [18], which targets data visualization and among other things, can be used to compress data. Data is represented at all resolutions. The user can specify different selections of data in order to meet space, time, and accuracy constraints in the context of visualization. In AC, we select only the data needed to meet the specified accuracy constraints, and we employ a multidimensional array to organize the data, a representation that supports the re-use of existing data analysis code more readily than a tree. Adaptive coarsening contains built-in heuristics for handling numerical properties specific to the solutions of partial differential equations, for example, anisotropic and mixed interpolation schemes.

We are currently implementing a framework that supports the data representation employed by Adaptive Coarsening, enabling users to apply their existing analysis algorithms to the patched-based compressed data product. This effort is part of project involving a computational database currently funded by the NSF CDI program. We will report on this work in the future.

Acknowledgments

The authors gratefully acknowledge the assistance of Sutanu Sarkar, Kyle Brucker, and Hieu Pham (UCSD MAE Department) for providing in insight and access to the 3D shear flow datasets; Truong Nguyen (UCSD ECE Department) for his advice in conducting the wavelet compression experiments; and Elio Damaggio for his contribution to developing 2D anisotropic sampling.

This work was supported in part by NSF contract ACI9619020 (National Partnership for Advanced Computational Infrastructure (NPACI)) and in part by the University of California, San Diego. The cluster was obtained with the support of NSF contract ACI0326013. Didem Unat was supported by a Center of Excellence grant from the Norwegian Research Council to the Center for Biomedical Computing at the Simula Research Laboratory.

References

[1] P. Ratanaworabhan, J. Ke, and M. Burtscher, "Fast lossless compression of scientific floating-point data," in *Proc. Data Compression Conf.*, 2006.

[2] E. Engelson, D. Fritzson, and P. Fritzson, "Lossless compression of high-volume numerical data from simulations," in *Proc. Data Compression Conf.*, pp. 574–586, 2000.

[3] J. Wilhelms and A. V. Gelder, "Octrees for faster isosurface generation," *ACM Trans. Graph.*, vol. 11, no. 3, pp. 201–227, 1992.

[4] L. A. Freitag and R. M. Loy, "Adaptive, multiresolution visualization of large data sets using a distributed memory octree," in *Proc. SC99*, November 1999.

[5] F. Aràndiga and R. Donat, "Nonlinear multi-scale decompositions: the approach of a. harten," *Numer. Algor.*, vol. 23, pp. 175–216, 2000.

[6] J. Roerdink and M. Westenberg, "Wavelet-based volume visualization," Tech. Rep. 98-9-06, Univeristy of Groningen, 1998.

[7] R. Belfor, M. Hesp, R. Lagendijk, and J. Biemond, "Spatially adaptive subsampling of image sequences," *IEEE Trans. Image Proc.*, vol. 3, no. 5, pp. 492–500, 1994.

[8] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *J. Comput. Phys.*, vol. 53, pp. 484–512, March 1984.

[9] M. J. Berger and P. Colella, "Local adaptive mesh refinement for shock hydrodynamics," *J. Comput. Phys.*, vol. 82, pp. 64–84, May 1989.

[10] T. M. Shafaat, "Context dependent compression using adaptive subsampling of scientific data," *M.S Thesis, Kungliga Tekniska Högskolan, Stockholm, Sweden*, 2006.

[11] T. M. Shafaat and S. B. Baden, "A method of adaptive coarsening for compressing scientific datasets," in *Applied parallel computing: State-of-the-Art in Scientific and Parallel Computing, 8th Intl. Workshop, Proc. PARA '06", Umeå, Sweden, Jun. 2006* (B. Kågström, E. Elmro, J. Dongarra, and J. Wasniewski, eds.), New York: Springer-Verlag, 2007.

[12] C. R. Schroeder, "Adaptive coarsening: Simple, effective floating-point compression (poster)," in *Proc. SC06*, November 2006.

[13] C. R. Schroeder, "Adaptive coarsening in 2d (class project for cse 262)," *Univ. of California, San Diego, Dep. of Computer Sci. and Engineering*, 2006.

[14] S. Basak and S. Sarkar, "Dynamics of a stratified shear layer with horizontal shear," *J. Fluid Mech.*, vol. 568, pp. 19–54, 2006.

[15] "Gnu scientific library." http://www.gnu.org/software/gsl.

[16] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," tech. rep., Argonne National Laboratory, Argonne, IL, 1997. http://www.mcs.anl.gov/mpi/mpich.

[17] H. Hoppe, "Progressive meshes," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 99–108, ACM, 1996.

[18] J. Wilhelms and A. V. Gelder, "Multi-dimensional trees for controlled volume rendering and compression," in *VVS '94: Proceedings of the 1994 symposium on Volume visualization*, (New York, NY, USA), pp. 27–34, ACM, 1994.