Scalable Computation of Distributions from Large Scale Data Sets

Abon Chaudhuri[1], Teng-Yok Lee[1], Bo Zhou[3], Cong Wang[3], Tiantian Xu[3], Han-Wei Shen[1], Tom Peterka[2], and Yi-Jen Chiang[3]

[1]The Ohio State University* [2]Argonne National Laboratory† [3]Polytechnic Institute of NYU‡

Abstract

As we approach the era of exascale computing, the role of distributions to summarize, analyze and visualize large scale data is becoming more and more important. Since histograms continue to be a popular way of modeling the underlying data distribution, we propose a scalable and distributed framework for computing histograms from scalar and vector data at different levels of detail required by various types of analysis algorithms. We present efficient parallel techniques for histogram computation from regular as well as rectilinear grid data. We also study a technique called cross-validation to estimate the quality of computed histograms as a model of the actual data distribution. We parallelize cross-validation in a scalable manner to support histogram evaluation and selection of histogram parameters such as number of bins. We also present our distributed software framework for supporting science applications which require large scale distribution-based data analysis. The presented case studies highlight how the proposed algorithms and the related software benefit information theoretic and other distribution-driven analysis.

1 Introduction

Distributions play a crucial role in numerous analysis and visualization techniques. For example, conventional methods such as transfer function design for volume rendering are often guided by locally and globally computed distributions [10, 13]. In recent years, as we approach the era of exascale computing, the role of distributions to effectively manage, analyze, and visualize data is emphasized even more. On one hand, certain types of simulations such as stochastic or ensemble simulations produce data in the form of distribution. On the other, emerging algorithms driven by uncertainty [26] or information theory [28, 4] often require distributions of many types to be computed at various scales from the data. Since these methods focus more on different analysis using the distributions, computing distributions at multiple scales efficiently and accurately remains a challenging problem.

Distributions are either represented by a set of parameters or as a histogram. Histograms are widely used for scientific data because they are able to produce a meaningful summary even when the data cannot be modeled by any known type of distribution. However, the process of converting large amounts of raw data into histograms is challenging. First, the method of computing histogram from data may vary based on the data type and its spatial organization. For example, defining and partitioning a range for vector quantities may not be trivial. Also, scientific data are available in different type of grids, including regular grids, rectilinear grids, curvilinear grids, or even unstructured grids. Second, depending on the analysis task, histograms may be needed at different resolutions (in terms of number of bins) and at different levels of detail such as a global histogram for the entire data, a histogram for each data block, or even a point-wise histogram for each voxel. To achieve this, one or multiple scans of the data may be required. Third, as histograms with different parameters, most importantly value range and number of bins, model the underlying data distribution with different levels of accuracy, the parameters should be carefully selected.

To address these issues, this paper proposes a framework for efficient computation of histograms for large-scale scientific data. The broader focus of this paper is on this distributed and scalable framework for computing histograms from both vector fields and scalar fields at various resolutions and level-of-details from large datasets. In particular, we identify a set of key challenges related to this and propose novel solutions to them. This includes an algorithm for computing histograms from vector directions in a scalable manner, an algorithm for computing histogram from rectilinear grid data, and a distributed method for analyzing accuracy of histogram representation.

To compute histograms at multiple resolutions from vector data based purely on vector directions, we adopt geodesic grid based spherical partitioning to develop a hierarchical parallel method. Moreover, we provide an efficient weighted vertex method for computing histograms from rectilinear grid data. We also utilize a statistical technique known as *cross-validation* to estimate the quality of a histogram as a model of the data and use this method to guide parameter selection for histograms. However, this technique involves multiple passes through the data which limits its use on real and large data. So we propose a scalable and distributed method of cross-validation to enable fast evaluation and parameter selection of large number of histograms computed from extreme scale data. Finally, we demonstrate that these techniques benefit information theory driven data analysis in large scale science applications and they can be useful for other types of distribution based analysis as well.

The remainder of this paper is organized as follows: Section 2 summarizes the related work. Section 3 explains distributed computation of histograms at different levels. Histogram computation for rectilinear grids is covered in Section 4. Distributed cross-validation is the focus of Section 5. Applications with case studies are discussed in Section 6, followed by a description of our software platform in Section 7. A detailed performance analysis in Section 8 is followed by conclusions in Section 9.

2 Related Work

Fast and parallel histogram computation is considered a pertinent problem in many different fields which extensively use histograms such as visualization, image analysis, computer vision and databases. An early work [8] on parallel image processing proposes a distributed algorithm for computing global image histogram by combining local ones. GPU-based parallelization of histogram computation is a complementary and well-studied problem [6, 19, 15]. Also, the wide use of mutual information, a joint

*The email addresses of Abon Chaurhuri, Teng-Yok Lee, and Han-Wei Shen are {chaudhua,leeten,hwshen}@cse.ohio-state.edu

†The email address of Tom Peterka is tpeterka@mcs.anl.gov

‡The email addresses of Bo Zhou, Cong Wang, Tiantian Xu, and Yi-Jen Chiang are {bzhou02, cwang05, txu04}@students.poly.edu and yjc@poly.edu, respectively.

distribution based similarity measure, has motivated research on its GPU-based parallel computation [21, 12]. The limitation of these GPU-based approaches are that they are designed for in-core computation, but not for distributed environment. Also, they are employed to compute entropy or mutual information for the entire data domain, not for per location computation. In contrast, we focus on processing extremely large data on a distributed platform, and computing histograms at various levels of detail.

Distribution-based analysis of scientific data is an emerging area to be further explored in coming years. The strategy of hierarchical block-based processing enables comparing and grouping of blocks based on distributions [9]. Another recent work [26] shows how to visually convey the uncertainty associated with histograms computed at various levels of details. Martin and Shen [14] propose histogram spectra, a collection of histograms defined over a range of sampling frequency, to achieve interactive level-of-detail rendering of large scale time-varying data. As these techniques focus on different types of analysis, all of them underline the importance of a scalable technique for computing histograms.

Various techniques are available for determining histogram parameters. Sometimes, heuristic-based formulae such as $\sqrt{N}$ where N is total number of observations, or Sturges' formula [20] are used to estimate number of bins. Some other techniques such as Doane's formula [20], Scott's normal reference rule [20] and Freedman and Diaconis rule [7] rely on the basic statistics of the data. This paper studies and parallelizes cross-validation [17, 3, 24, 22], a parameter selection method which does not assume availability of any statistic of the data other than the histogram itself. Unlike the other methods which return an optimal number, cross-validation evaluates the parameter space and returns a goodness of fit measure also for the sub-optimal parameters. A recent work by Shimazaki et al [23] optimizes a cost function to objectively select bin width for a specific type of histogram used in neurophysiology research.

3 Distributed Histogram Construction

Analysis algorithms process data at different levels of detail. For example, one algorithm may analyze blocks of size 8^3, while another may operate on blocks of 16^3. Similarly, different algorithms may work with histograms of different resolutions (in terms of number of bins). Our objective is to compute a set of histograms which can produce many other histograms required to support a wide variety of algorithms.

3.1 Histogram Computation

The core operation in histogram computation is a mapping from the n-dimensional data values to the bin IDs, denoted by $f : R^n \rightarrow [1, K]$, where each of the K bins contains one value, the frequency. The proposed distributed computation is independent of the exact method used for this mapping.

3.1.1 Histogram from Scalar Data

In general, computing a histogram from scalar observations is relatively straightforward. Given the range of scalar values, denoted by $[m, M]$, and a bin width h, the bin ID I for a scalar s can be simply obtained by: $I = \lfloor \frac{s-m}{h} \rfloor$. If the value range is not known, an additional pass through the data is required to obtain that. In distributed computation, each block can compute its own range, and then the blocks can communicate among themselves to determine the global range.

3.1.2 Histogram from Vector Data

Computing a histogram from a vector field is challenging, since there is no standard way to specify and partition the value range. One strategy is to use some related scalar quantity such as vector magnitude for the histogram computation. But to compute histograms purely based on vector directions, a suitable mapping from the vector space to histogram bins is needed. In 3D (2D), a vector can be defined in terms of its spherical (polar) coordinates, so the surface (circumference) of a unit sphere (circle) represents the domain of vector directions. So, the problem of mapping reduces to partitioning a unit sphere into patches of equal area. Given that partitioning, each vector direction is assigned a bin based on which patch it intersect with (Figure 1a).

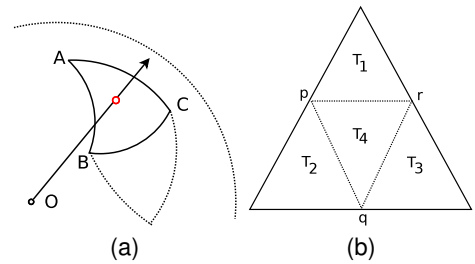


Figure 1: **(a)** Bin assignment of vectors based on spherical partitioning.. **(b)** Recursive subdivision of a triangle patch during geodesic grid construction.

Xu et al. [28] achieves this by adopting Leopardi's algorithm [11], which partitions a sphere into a specified number of quadrilateral patches of nearly equal area. However, this algorithm is not quite suitable for computing a histogram at multiple resolutions, because there is no guaranteed correspondence between the patches at resolution K and those at $2 \times K$.

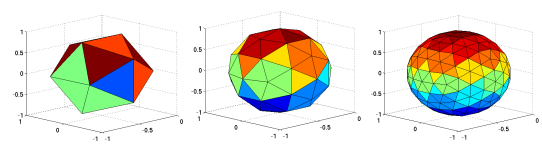


Figure 2: Construction of geodesic grid at different resolutions (Number of triangle patches are 20, 80 and 320 from left to right).

To address the issue, we utilize a *geodesic grid* to partition the spherical space, as approach widely used by geoscientists to model the earth's surface [18, 27]. The geodesic grid starts from an icosahedron (or other polyhedron) inscribed in a unit sphere and recursively tessellates it into finer ones. The tessellation of a triangular face (Figure 1b) into four equal area triangles is achieved by creating four vertices at the midpoints of the triangle edges. After tessellation, the newly created triangles are projected onto the unit sphere. This implies that, at any resolution, each triangular face of the polyhedron has nearly equal area and can be used as a bin. Moreover, this subdivision enforces a parent-child relation between the bins of the consecutive levels. By combining the frequencies of the finer patches, it is possible to compute the frequency of the parent patch without accessing the data. Hence, this algorithm enables computation of histograms at multiple resolutions with one pass through the data.

After tessellation, each vector needs to scan through all the patches to find the one it intersects with. To accelerate this bin assignment stage, a pre-computed 2D lookup table of bin IDs at sampled spherical coordinates (ϕ, θ) is used.

3.2 Distributed Histogram Computation

In a distributed environment, *blocking* of the data and *assignment* of a group of blocks to each processor precedes the actual histogram computation.

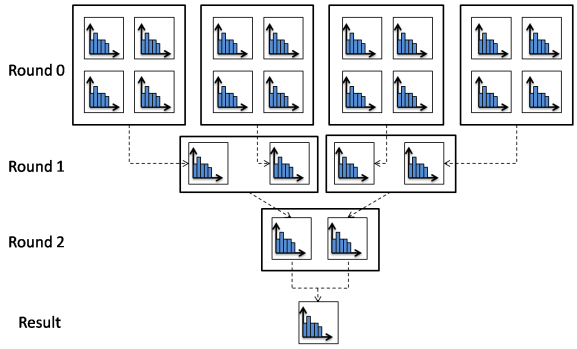


Figure 3: Distributed block-wise and global histogram computation by multi-round reduction.

3.2.1 Multilevel Histogram Computation

Analysis tasks may require histograms at different levels of detail such as voxel level, block levels of varying sizes and the global level containing the entire data. The multi-level computation is possible with one scan of the data due to the following property: The histogram of a larger region can be accurately constructed from the histograms of its partitions, if they use the same resolution. Simple addition of the frequencies of corresponding bins is sufficient. This indicates that the histogram of a block of dimension $2B \times 2B \times 2B$ can be constructed by combining the histograms of the $B \times B \times B$ blocks contained in it.

Our framework supports histogram computation at three different levels: block-wise, global and local. For *block-wise histogram* mode, the data is partitioned into blocks of size corresponding to the finest level of detail needed (say 16^3). Then a histogram per block is computed and stored. During analysis, the stored block histograms can be read back and iteratively merged in parallel until the histograms of the some required level are obtained (Figure 3). Merging the histograms all the way down to a single one produces the *global histogram*. We also compute *point-wise* or *voxel-wise* or *local* histogram, which is the finest level of detail possibly needed. Ideally, the local histogram at each point or voxel is the histogram based on the data point itself. Such histograms can be combined together to obtain block histograms of any size. However, we compute a more general quantity at each point based on a neighborhood of size δ (represented by a 3D sub-volume of size $(2 \times \delta + 1)^3$). This neighborhood is slid across the data domain in all dimensions (Figure 4). We compute this because of its use it information theoretic analysis (to be explained in Section 6). When data is distributed among processors as blocks, the points at the boundary of a block need data from outside the block to construct the neighborhood. This is why each block is read with extra ghost layers of width δ in all dimensions. Alternately, the neighboring blocks can exchange ghost layers among themselves before computation starts.

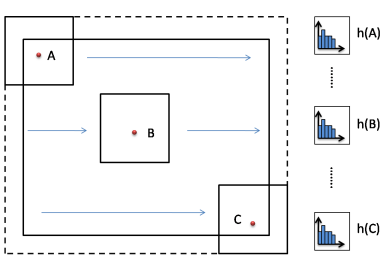


Figure 4: Local histogram computation at each point within a block.

3.2.2 Multi-resolution Histogram Computation

Also, histograms of different resolutions (number of bins) may be required for different tasks and datasets. To our benefit, a low resolution histogram can be accurately constructed from its higher resolution counterpart. A histogram of $\frac{K}{t}$ bins can be accurately computed by adding up the frequencies of all the consecutive bin groups of size t of a histogram of K bins. By applying this operation progressively, $h_{K_p}(R)$ for a given region R can be accurately constructed from $h_{K_1}(R)$, if $K_p = t^p \times K_1$. Hence, in practice, a high-resolution histogram is computed at each level. As needed, they are first merged to form the histogram(s) of the desired level, and then coarsened by grouping the bins to the desired resolution.

4 Histogram Computation on Rectilinear Grids

In this section we extend our consideration of histogram computation to rectilinear grids. In particular, we consider how to compute the local histogram at each vertex of a rectilinear grid. At each vertex v, we define a neighborhood box of size t (a 3D cube centered at v with side length $2t$ in each dimension), and construct the histogram within the box. (All methods presented here can also be used directly at block level, by considering the neighborhood box as a block.) Unlike regular grids where vertices represent a uniform sampling over the volume and have the same weight, in rectilinear grids, due to the rectilinear structure we cannot apply the regular-grid approaches since they do not guarantee enough accuracy (i.e., data samples are not properly weighted). So we present three methods suitable for rectilinear grids and an optimized version of the best among the three.

Method 1: Sampling: For a given neighborhood box, we uniformly sample all the cells contained in it with sufficient numbers of points. For cells on the box boundary that partially intersect the box, we only include those sample points that are inside the box. For each sample point, we interpolate to obtain the data value, find out which histogram bin the value belongs to, and assign the weight of this sample point to that bin. The weight of a sample point is the volume of the cell divided by the number of sample points in the cell. This method offers a trade-off between speed and accuracy, controlled by the sampling density.

Method 2: Contour Spectrum: This method is based on contour spectrum [2], which is restricted to *scalar* data over a *tetrahedral* mesh. It computes the exact isosurface area at each scalar value from infinite and continuous sample points over each tetrahedron, and thus the resulting histogram is accurate and free from sampling errors. To apply this technique to rectilinear grids, we split each rectilinear cell into 5 tetrahedra, and calculate the volume distribution of each tetrahedron into histogram bins using contour spectrum by integrating the resulting B-spline function over the bin ranges covered. For each cell on the neighborhood-box boundary, we only consider the part within the neighborhood box, split this part into 5 tetrahedra, and apply the same process.

This method is usually more accurate than sampling without the need to worry about the sampling density. However, different ways of splitting a rectilinear cell might lead to quite different results.

Method 3: Weighted Vertex: This method resembles the method of computing local histograms on regular grids (Section 3.2.1). For each vertex, we consider the grid vertices inside its local neighborhood box. In addition, the intersection points between the box boundary faces and the grid edges, as well as the 8 corners of the box, are included. Data values at these additional points on the box boundary, namely the *pseudo-vertices*, are obtained via interpolation. The pseudo-vertices and the grid vertices inside the box form the *box vertices*. Each box vertex is assigned to a bin based on its data value, and its

vertex weight is added to the bin. The volume of the axis-aligned bounding box of the midpoints of the 6 edges incident to a box vertex determines its weight. For pseudo-vertices, the weight only includes the portion inside the box (Figure 5).

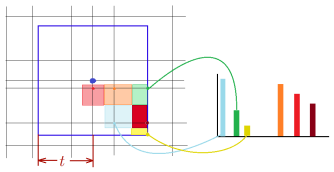


Figure 5: A 2D illustration of the weighted vertex method. The green and yellow points on the box boundary are examples of pseudo-vertices.

The execution time of this method is proportional to the number of box vertices. If the *average* number of box vertices is $k \times k \times k$, then the overall runtime for computing all local histograms is proportional to $O(k^3)$. Results in Section 8.2 indicates that this is the fastest method among the three presented so far.

As for accuracy, weighted vertex would perform badly if the data values between adjacent grid vertices vary dramatically, which typically does not happen since a reasonable rectilinear grid uses higher grid resolution in such feature-rich regions. Therefore weighted vertex has a good accuracy for typical rectilinear grids. In Fig. 6 we show the local entropy results of the three methods on a representative 2D scalar field, where the difference of scalar values between neighboring vertices is roughly the same over the whole mesh. We can see that the weighted vertex method can produce a local entropy result similar to those of the other two methods. In other words, it achieves a similar good accuracy with the fastest running time, and should be the method of choice.

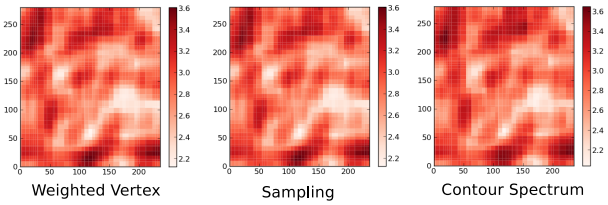


Figure 6: Color maps of local entropy values on a representative 2D scalar field.

Efficient Algorithm for Weighted Vertex: Sliding Window: We can further optimize the efficiency of the weighted vertex method. Since the neighborhood boxes of two neighboring vertices share many common vertices, local histogram of a vertex can be updated from that of its preceding neighbor. We first look at the easier case of regular grids (Figure 7(a)). To build the histogram for vertex B given the histogram of A, we only need to remove the contribution of vertices $a, b, \cdots, e$ (in the *back-slice plane*), and add the contribution of $a', b', \cdots, e'$ (in the *front-slice plane*) into the bins. All vertices in common for A and B do not need to be recomputed. This updating accelerates the sliding neighborhood approach described in Section 3.2.1. Given a neighborhood box of resolution k^3, this brings down computing cost to $2k^2$ from k^3.

The main idea is the same for rectilinear grids. We consider the general case where the boundaries of the neighborhood box are not on the grid lines (see Fig. 7(b)). When we slide the window center from A to the next grid point B, we need to add two new slices S_1, S_4 which are the front and back slices of the neighborhood box of B (type (1) updates), and remove two old slices S'_1, S'_4 which are the front and back slices of A (type (2) updates). Also, we need to

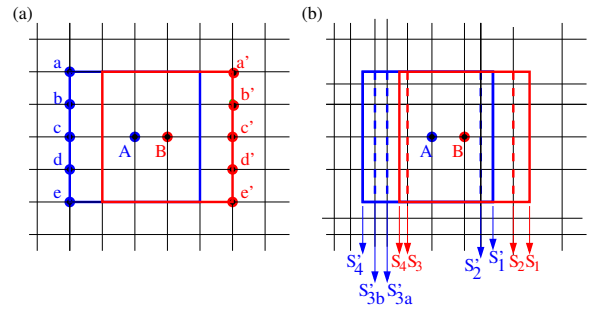


Figure 7: Sliding window: (a) for regular grids, (b) for rectilinear grids. The blue solid box is for A and the red solid box is for B.

update two slices S'_2, S_3 due to weight changes (type (3) updates): S'_2 is the slice on the grid line right after the front slice of A; the weights were originally constrained by this front slice but now not any more. Similarly, S_3 is on the grid line right before the back slice of B, and the weights are now newly constrained. Moreover, we need to perform the following type (4) updates: we add new slices on grid edges such as S_2, and remove old slices on grid edges such as S'_{3a} and S'_{3b}. Overall, there are four types of updates.

Note that there can be an *unbounded* number of type (4) update slices in one sliding step due to the rectilinear structure. However, in the entire window sliding process, each grid line will enter the box once and then leave the box once. Thus, each grid line causes one slice of addition and one slice of removal. Also, the total number of grid lines equals to the total number of window-sliding steps, and thus we have one slice addition and one slice removal per window-sliding step on an average for type (4) updates. Since there are 2 slices of work in each of types (1)-(3) updates, the total work per window-sliding step is 8 slices on an average.

Suppose the *average* number of box vertices is $k \times k \times k$, then the average work per histogram is $O(k^2)$, as opposed to $O(k^3)$ in the original weighted vertex method. Typically this improvement is significant, especially for large datasets and/or large box sizes.

Parallel Computation: For all the methods presented, we parallelize them by concurrently performing the sequential computation on each partition of the grid assigned to each processor after getting the ghost layers through communication, if needed.

5 Distributed Parameter Selection of Histograms

A histogram is defined by two parameters: value range and bin width (equivalent to number of bins). While the bin width can be non-uniform, here we restrict the discussion to uniform bin width. Since the histogram represents an estimate of the probability density function (PDF) of an unknown distribution, there is no straightforward way to know the parameter values at which the histogram best approximates that PDF. The range of the data values can guide the histogram range and start point. But there is no clear indicator for choosing the histogram resolution. This is, however, important because at a suboptimal resolution, histogram modality may change and important features may be lost. On the other hand, at a very high resolution, the histogram may over-fit the sampled data points.

5.1 Cross-Validation

In general, given a model with unknown parameters and a dataset, a model evaluation technique called *cross-validation* is used to assess how well the model fits the dataset. This technique is also applied to evaluate how well a histogram models the underlying distribution of the data from which it is computed. Since the actual parameters of the data distribution are not known, a true validation of the histogram is not possible.

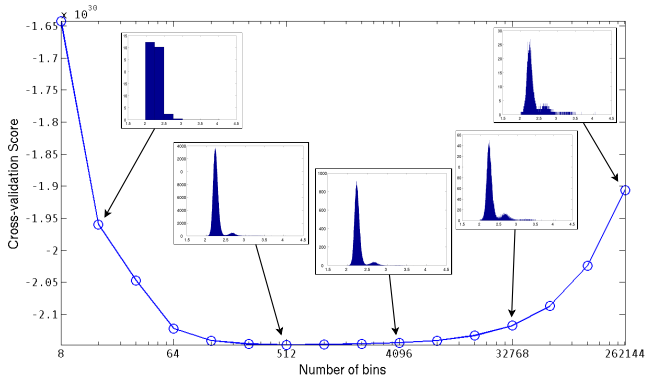


Figure 8: Cross-validation of a 64^3 block from FLASH dataset over a wide range of bin numbers.

Here we provide a brief account of cross-validation [17, 3, 24, 22] as histogram parameter estimator. For a given sample x, if $f(x)$ denotes its true density and $\hat{f}_h(x)$ denotes the density estimated by a histogram of bin width h, then integrated square error (ISE) over all samples serves as a measure of closeness. ISE between the actual and the estimated distribution is given by:

$$\begin{aligned} ISE &= \int \left(\hat{f}_h(x) - f(x)\right)^2 dx \\ &= \int \hat{f}_h(x)^2 dx - 2\int \hat{f}_h(x) f(x) dx + \int f(x)^2 dx \end{aligned} \quad (1)$$

Hence, the goal of parameter (h) selection is to find h which minimizes the ISE. The first term in its expansion (Equation 1) is computable from the histogram. The third term, although unknown, is independent of h and can be regarded as a constant. Estimation of the second term is crucial in estimating ISE for a given h. As the second term essentially means the average of the estimated density function $\hat{f}(x)$, Rudemo [17] uses a leave-one-out strategy to generate multiple histograms from a set of given samples, and computes their average as the second term. Interestingly, Rudemo [17] provides a closed form *histogram estimator*, denoted by Q, which allows to estimate the ISE for a data with n observations and K bins with fixed bin width h. It is defined as:

$$Q = \frac{2}{(n-1)h} - \frac{n+1}{n^2(n-1)h}\sum_K {n_k}^2 \quad (2)$$

We use this formula to provide an assessment of the histograms computed with different resolutions from a data set. Figure 8 presents the cross-validation risk scores for the global histogram of a 64^3 block from FLASH simulation data (see Section 6). The scores suggest that 512-4096 is the optimal number of bins for this data. Observing the histograms in insets, we can see that the two modes of the histogram are prominently visible roughly within this range. At a coarser resolution, the smaller right mode disappears and at very high resolutions, noisy peaks start to appear.

5.2 Parallelized Cross-Validation

Serial cross-validation can be quite prohibitive on large datasets over a large range of histogram resolutions because each iteration, representing a resolution, requires a complete scan of the data. Running different iterations in parallel is not scalable since the number of iterations gets limited by the size of the parallel grid. In this paper, we propose two strategies for scalable parallel cross-validation.

Multiple Data Access Method: After loading blocked data in parallel, the histogram of each block is computed for a wide range of histogram resolutions. Then, for each resolution, the block histograms are merged in parallel to compute the respective global histograms (Figure 3) in a central processor. The central processor then computes the cross-validation risks of all the global histograms, plots them and finds the minimum (or minima). This method has a limitation: even though it parallelizes each scan of the data, it scans the data as many times as the serial version.

Single Data Access Method: The above method can be accelerated even more by restricting the data access to one. In this version, the range of histogram resolution, spanning from N_m to N_M, is sampled based on a geometric progression. Starting with the highest histogram resolution N_M, following iterations examine $\frac{N_M}{2}$, $\frac{N_M}{2^2}$ and so on. The algorithm stops at N_m after p iterations where $N_M = 2^{p-1}N_m$. Choosing histogram resolutions in this way allows us to compute the histograms without repeated scanning of the data (Section 3.2.2). The acceleration comes at the cost of sparse sampling of the histogram resolution range. While using this method, the suggested optimal bin ID is the power of 2 closest to the actual number which is reasonably accurate in most cases.

If computation time is not a major concern, a shorter range of resolutions can be thoroughly scanned (using first method) around the approximate resolution returned by the second.

6 Applications

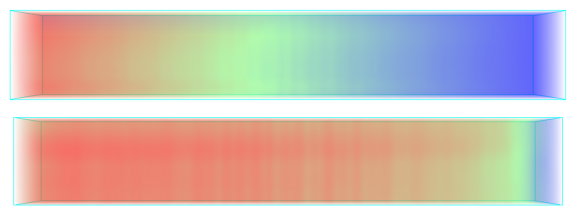


Figure 9: Fluid pressure along pipe (top) and respective entropy field (bottom) from a Nek5000 scalar data.

6.1 Information Theory Based Science Applications

Information theory based visualization techniques often require Shannon's entropy and related metrics computed at different levels of detail from the data. For example, streamline placement can be guided by pointwise entropy [28], isosurfaces can be compared based on the global mutual information between them [4]. Computation of all these metrics require the data distribution (or joint distributions) at the appropriate level of detail. Hence, the proposed framework for histogram computation is directly applicable to scalable computation of Shannon's entropy and related metrics in global, blockwise or pointwise mode.

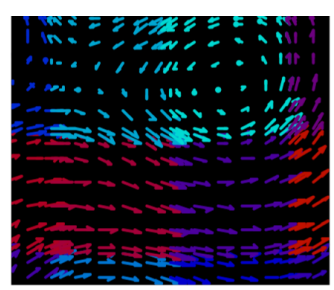
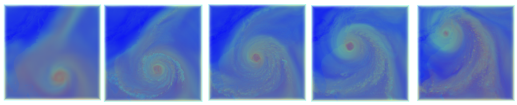


Figure 10: A few blocks from a Nek5000 vector field. Each glyph is colored based on the entropy of its block.

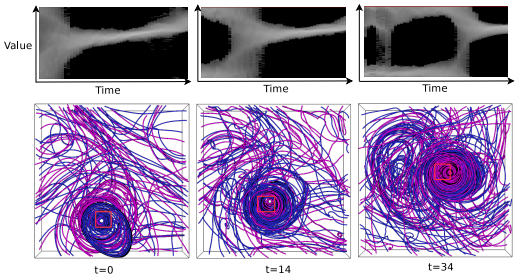
In particular, two computational science codes, Nek5000 and FLASH, have motivated this research. Nek5000 is a Navier-Stokes solver used for the simulation of fluid flow, convective heat and species transport, and magnetohydrodynamics in 2D and 3D. It is based on a spectral element solver [5] and features an unstructured mesh

consisting of high-order hexahedral cells. FLASH is an adaptive mesh code used primarily for astrophysics and cosmology [25]. Here we present an example from Nek5000 which simulates fluid flowing though a straight pipe. We are able to read and analyze the data in situ in this example. Figure 9 presents the fluid pressure field (top). Since pressure varies linearly along the length of the pipe, the corresponding point-wise entropy field (bottom) remains more or less constant at a high value. In Figure 10, a portion of a vector field produced in another Nek5000 example is shown. Here the glyphs representing vectors have been color coded using the entropy of their respective blocks. This helps to differentiate between blocks of varying complexity. A block containing a vortex is present (in bright blue) in the segment shown.

6.2 Spatio-Temporal Analysis



(a) Isabel hurricane dataset with a moving vortex.



(b) Time histograms of three blocks and their actual locations.

Figure 11: A case study showing feature localization in spatio-temporal domain using block-wise time histograms.

We present another application to highlight the importance of histogram based analysis beyond information theory. We can employ our framework to fast compute time histograms [1] from large scale data which are ordered stacks, growing horizontally, of histograms computed from multiple time steps. When computed globally, they capture the dynamic behavior of the data over time. However, global time histogram cannot reflect a feature's behavior in the spatial domain. Figure 11 shows that time histograms computed at block levels can serve as visual signatures of a features behavior over space and time.

Equidistant time steps taken from the Isabel hurricane ($500 \times 500 \times 500$) simulation (Figure 11a) shows a moving vortex. In general, a uniformly spread histogram (represented by a bright vertical strip on a time histogram) should indicate the presence of a vortex or similarly turbulent region [28]. Hence, given the time histogram of a block, band(s) formed by such adjacent bright strips signifies the duration(s) for which the moving vortex has some overlap with the block. On the other hand, given a time step, the user can quickly scan though various time histograms to narrow down on the blocks where the vortex appears. Figure 11 presents three block-level (25^3) time histograms (top row) computed from the wind velocity fields and the corresponding blocks (highlighted by red boxes on the bottom row). The time histograms clearly indicate the hurricane's presence in the three blocks in three different time steps.

7 Software

We are developing a software framework called *Information Theoretic Library (ITL)* to extend the research benefits to scientists and developers of science applications. Figure 12a illustrates the capability of the software. The first stage converts various types of data into histograms of desired level and resolution. Cross-validation is optionally performed while computing histograms.

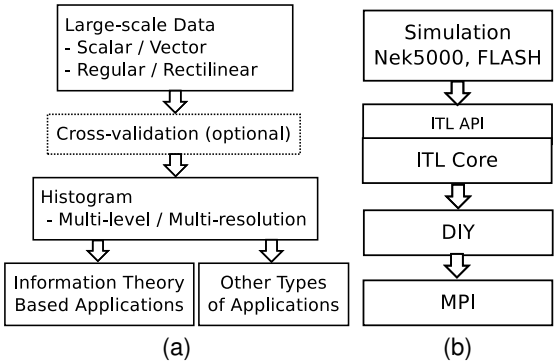


Figure 12: (a) Data analysis tasks supported by our software framework. (b) ITL library structure and in situ usage in a simulation.

ITL is a parallel C/C++ based library containing a set of lightweight tools, which can be plugged into a simulation code for in situ analysis, or can be used as a stand-alone tool for post-processing of simulation results. The library itself has two main components, *ITL API* and *ITL core* (Figure 12b). ITL API provides necessary wrappers to make ITL functionalities compatible with a simulation code. ITL core implements the main data-to-histogram and histogram-to-entropy computation modules. The parallelism is driven by an MPI-based library called DYI [16]. The end users, who are the owners and maintainers of simulation codes, need to access and modify only the ITL API to integrate their code to ITL. ITL core and DIY are transparent to the users.

8 Performance

This section shows the performance scalability of our implementation. The timings are measured on the supercomputer *Surveyor* at the Argonne Leadership Computing Facility. *Surveyor* is an IBM Blue Gene/P supercomputer with 1024 quad-core processors. All tests are conducted in smp mode, which runs only one core at each processor. Since every communication takes place between different processors in this mode, the worst possible overhead due to communication is captured. The testing datasets are listed in Table 1. Both *Plume* and *Nek* are regular grid vector data. We use the vector magnitude per voxel to form a corresponding scalar field for testing. This allows us to compare the performances between scalar and vector fields because the data resolution is the same. *Rect-S* and *Rect-L* are rectilinear scalar fields. All figures of the performances are plotted in logarithmic scale for both x and y axes.

8.1 Histogram Computation on Regular Grids

Figure 13 analyzes the performances of the three supported modes, global, block-wise and local, of histogram computation for regular grids (Section 3.2.1).

Figures 13 (a) - (d) list the performance of block-wise histogram computation. Since the number of blocks per processor can be specified by the user, we also measure the performance by varying this parameter. We observe that the performances are close regardless of the number of blocks.

Figure 13(e) lists the performance of global histogram computation for both scalar and vector fields from Nek and Plume dataset.

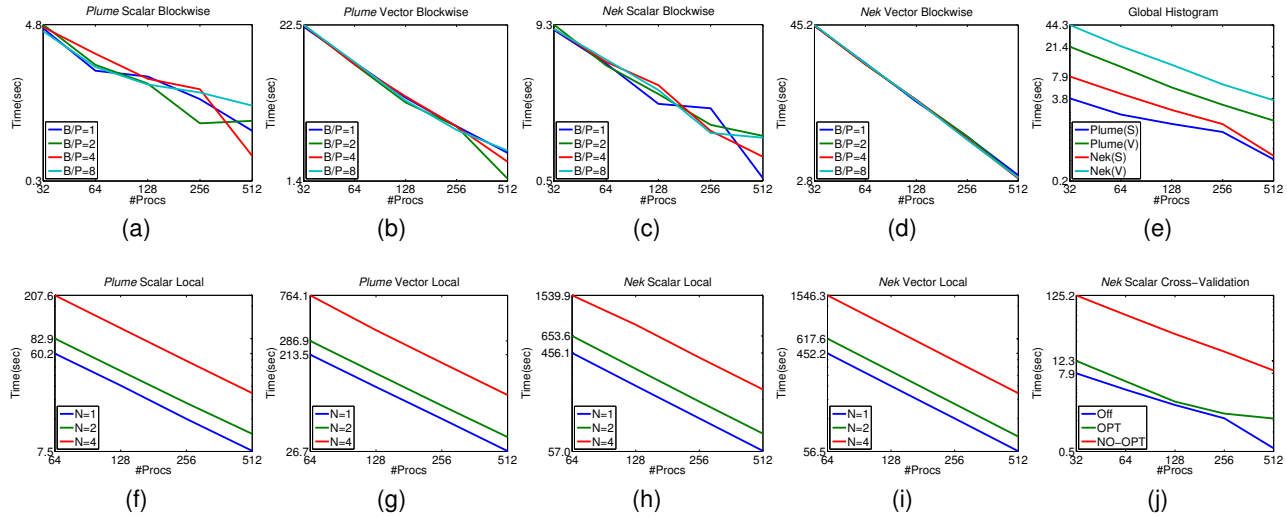


Figure 13: Performance of histogram computation. 360 bins are used for all tests. (a)-(d). Performance of block-wise histogram computation. B/P in the legend means the number of blocks assigned to each process. (e) Performance of global histogram computation. Legends S and V mean scalar field and vector field respectively. (f)-(i). Performance of local histogram computation. N represents neighborhood size. (j). Performance of cross validation for the scalar field of *Nek*. Symbols OFF, OPT, and NO-OPT represent no cross validation, optimized cross-validation, and cross-validation without optimization respectively. Both x and y axes of all figures are in logarithm scale.

Table 1: Testing Datasets.

Dataset	Type	Grid	Resolution	Size
Plume	Scalar Vector	Regular	$504 \times 504 \times 2048$	2.1GB 6.2GB
Nek	Scalar Vector	Regular	$1024 \times 1024 \times 1024$	4.0GB 12GB
Rect-S	Scalar	Rectilinear	$704 \times 540 \times 550$	2.4GB
Rect-L	Scalar	Rectilinear	$1408 \times 1080 \times 1100$	19GB

The computation time for vector fields is at least 5 time larger than that for scalar fields. Computation of global histograms also requires data communication to merge the histograms from all processors (Section 3.2.1). Communication timing is not shown here, since it always remains less than 1 second in all performed tests.

Figures 13 (f) - (i) show the performance of local histogram computation under different neighborhood sizes. Compared to global and block-wise computations, the computation time is larger, and increases with neighborhood size. However, scalability is maintained regardless of the neighborhood size. This implies that our implementation has good scalability when the computation load is heavy. Better scalability of vector datasets in block-wise and global computation (Figure 13 (b),(d) and (e)) also supports the claim.

To evaluate the performance of cross-validation, we measure its performance for global histogram computation from *Nek*. Figure13 (j) shows the performance without cross validation run with 360 bins (*Off*), non-optimized multiple data access cross validation run through 1000 to 16000 bins in steps of 1000 (*NON-OPT*), and optimized single data access cross validation run through 2 to 65535 in powers of 2 (*OPT*) (Section 5.2). The scalability for non-optimized cross validation is better than the other two, due to the higher computation overhead of computing histograms at several resolutions. The optimized cross-validation is at least 9 times faster than the non-optimized one except on 512 processors (5x).

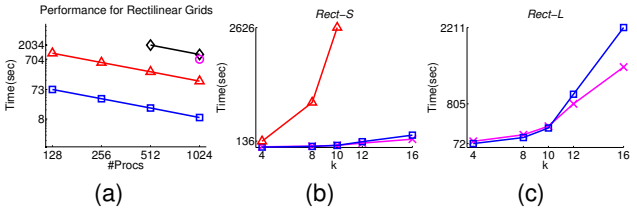


Figure 14: Performance analysis of proposed methods for rectilinear grids. **(a)** Runtime and scalability for sampling with $d = 0.008$ ($\triangle$) and $d = 0.004$ ($\diamond$), weighted vertex ($\square$), and contour spectrum ($\bigcirc$). **(b)-(c)** Performance under different values of k (where the *average* number of box vertices is k^3) for sampling with $d = 0.008$ ($\triangle$), weighted vertex ($\square$), and sliding window ($\times$). Both x and y axes in (a) are in logarithmic scale. In (b) and (c), only y axis is in logarithmic scale.

8.2 Histogram Computation on Rectilinear Grids

In Figure 14a, we present performance of the three proposed methods (sampling, contour spectrum and weighted vertex) on the *Rect-S* dataset. For the sampling method we use two sampling densities (with stepping distance $d = 0.008$ and $d = 0.004$ in each dimension). The neighborhood box size t used is 0.05 — the resulting *average* number of the *box vertices* (recall Method 3 in Sec. 4) was 4^3. The running time for computing local histograms of all vertices vary almost linearly with the number of processors for all three methods. The efficiency computed over the range of 128 to 1024 processors is 85%. Weighted vertex is generally much faster than the other two methods. Contour spectrum is the slowest (running out of time for all runs except for on 1024 processors). Also, the performance of the sampling method depends on the sampling density, as expected.

Another important parameter influencing run-time performance is the neighborhood-box size t. On both the *Rect-S* and *Rect-L* datasets, we vary t to be 0.05, 0.10, 0.14, 0.20, and 0.28 (the re-

sulting *average* number of the box vertices, as mentioned above, are k^3 with $k = 4, 8, 10, 12$ and 16 respectively) and run the sampling (with $d = 0.008$), weighted vertex, and sliding window methods on 1024 processors. The results in Figures 14b and 14c show that sampling is slower than the other two. It is too slow to be even compared with the other two on *Rect-L*. For the weighted vertex method, the running time grows fast with k. As mentioned in Sec. 4, sliding window reduces this k^3 units of work per histogram to $O(k^2)$. The advantage of sliding window becomes apparent for larger k and larger datasets.

It is interesting to observe that sliding window approach is a bit slower than weighted vertex for $k < 10$, the same speed at $k = 10$, and becomes much faster for $k > 10$. Considering the programming overheads, this confirms our analysis that sliding window has an average work of 8 slices per sliding operation (see Sec. 4).

We remark that when we run the tests with different numbers of bins, the resulting running time is the same for histogram computation by sampling, weighted vertex and sliding window methods, since the task of distributing each sample/vertex weight to a bin is the same. However, more bins would slow down the contour spectrum method as the cells tend to have more bins to integrate over.

9 Conclusion and Future Work

In this paper, we propose an efficient and scalable technique for computing large number of histograms from extreme scale scientific data organized in regular or rectilinear grid. We also apply cross-validation, a goodness of fit measure and parameter selection technique, concurrently on a large number of histograms. This research enhances information theoretic analysis of large simulation data and applies to many other distribution based analysis.

However, since the framework is still being developed, there is a plenty of scope for addition and improvement. To name a few, we intend to add support for curvilinear or unstructured grid data generated by many simulations. We also plan to deal with histograms with variable bin width. We are revisiting the histogram computation from vector data to take magnitude into consideration. Even though we have the basic support for computing kernel density estimation (the other form of representing distributions), we are working on optimizing and parallelizing it. Another major challenge to be addressed in future is to efficient storage and searching of large number of distributions.

Acknowledgements

This work was supported in part by NSF grant IIS-1017635, US Department of Energy DOE-SC0005036, Battelle Contract No. 137365, and Department of Energy SciDAC grant DE-FC02-06ER25779. We would like to thank program manager Lucy Nowell. The Nek5000 simulation dataset is courtesy of Aleks Obabko and Paul Fischer of Argonne National Laboratory and the FLASH dataset is courtesy of Paul Sutter.

References

[1] H. Akiba, N. Fout, and K.-L. Ma. Simultaneous classification of time-varying volume data based on the time histogram. In *EuroVis '06: Proceedings of the Joint Eurographics - IEEE VGTC Symposium on Visualization 2006*, pages 171–178, 2006.

[2] C. L. Bajaj, V. Pascucci, and D. R. Schikore. The contour spectrum. In *VIS '97: Proceedings of the IEEE Visualization 1997*, pages 167–ff., 1997.

[3] A. W. Bowman. An alternative method of cross-validation for the smoothing of density estimates. *Biometrika*, 71(2):pp. 353–360, 1984.

[4] S. Bruckner and T. Möller. Isosurface similarity maps. *Computer Graphics Forum*, 29(3):773–782, 2010.

[5] M. O. Deville, P. F. Fischer, and E. H. Mund. *High-Order Methods for Incompressible Fluid Flow*. 2002.

[6] O. Fluck, S. Aharon, D. Cremers, and M. Rousson. Gpu histogram computation. In *ACM SIGGRAPH 2006 Research posters*, 2006.

[7] D. Freedman and P. Diaconis. On the histogram as a density estimator:l2 theory. *Probability Theory and Related Fields*, 57:453–476, 1981.

[8] D. C. Gerogiannis, S. C. Orphanoudakis, and S. L. Johnsson. Histogram computation on distributed memory architectures. *Concurrency: Pract. Exper.*, 1(2):219–237, 1990.

[9] Y. Gu and C. Wang. Transgraph: Hierarchical exploration of transition relationships in time-varying volumetric data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2015 –2024, 2011.

[10] G. Kindlmann and J. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *VolVis '98: Proceedings of the IEEE Symposium on Volume Visualization 1998*, pages 79 –86, 1998.

[11] P. Leopardi. *Distributing Points on the sphere: Partitions, separation, quadrature and energy*. PhD thesis, The University of New South Wales, 2007.

[12] Y. Lin and G. Medioni. Mutual information computation and maximization using gpu. In *Proceeding of the Computer Vision and Pattern Recognition Workshop*, volume 0, pages 1–6, 2008.

[13] C. Lundstrom, P. Ljung, and A. Ynnerman. Local histograms for design of transfer functions in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1570 –1579, 2006.

[14] S. Martin and H.-W. Shen. Histogram spectra for multivariate time-varying volume lod selection. In *LDAV '11: Proceedings of the IEEE Symposium on Large Data Analysis and Visualization 2011*, pages 39 –46, 2011.

[15] C. Nugteren, G.-J. van den Braak, H. Corporaal, and B. Mesman. High performance predictable histogramming on gpus: exploring and evaluating algorithm trade-offs. In *GPGPU-4: Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-4, pages 1:1–1:8, 2011.

[16] T. Peterka, R. Ross, A. Gyulassy, V. Pascucci, W. Kendall, H.-W. Shen, T.-Y. Lee, and A. Chaudhuri. Scalable parallel building blocks for custom data analysis. In *LDAV '11: Proceedings of the IEEE Symposium on Large Data Analysis and Visualization 2011*, pages 105 –112, 2011.

[17] M. Rudemo. Empirical choice of histograms and kernel density estimators. *Scandinavian Journal of Statistics*, 9(2):pp. 65–78, 1982.

[18] R. Sadourny, A. Akio, and M. Yale. Integration of the nondivergent barotropic vorticity equation with an icosahedral-hexagonal grid for the sphere1. *Monthly Weather Review*, 96:351–356, 1968.

[19] T. Scheuermann and J. Hensley. Efficient histogram generation using scattering on gpus. In *I3D '07: Proceedings of the symposium on Interactive 3D graphics and games 2007*, pages 33–37, 2007.

[20] D. W. Scott. On optimal and data-based histograms. *Biometrika*, 66(3):pp. 605–610, 1979.

[21] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley. Parallel computation of mutual information on the gpu with application to real-time registration of 3d medical images. *Computer Methods and Programs in Biomedicine*, 99(2):133–146, 2010.

[22] S. J. Sheather. Density estimation. *Statistical Science*, 19(4):pp. 588–597, 2004.

[23] H. Shimazaki and S. Shinomoto. A method for selecting the bin size of a time histogram. *Neural Computation*, 19(6):1503–1527, 2007.

[24] C. J. Stone. An asymptotically optimal window selection rule for kernel density estimates. *The Annals of Statistics*, 12(4):pp. 1285–1297, 1984.

[25] P. M. Sutter and P. M. Ricker. Detecting Dark Matter-Dark Energy Coupling with the Halo Mass Function. *The Astrophysics Journal*, 687:7–11, 2008.

[26] D. Thompson, J. Levine, J. Bennett, P.-T. Bremer, A. Gyulassy, V. Pascucci, and P. Pebay. Analysis of large-scale scalar data using hixels. In *LDAV '11: Proceedings of the IEEE Symposium on Large Data Analysis and Visualization 2011*, pages 23 –30, 2011.

[27] D. L. WILLIAMSON. Integration of the barotropic vorticity equation on a spherical geodesic grid. *Tellus*, 20(4):642–653, 1968.

[28] L. Xu, T.-Y. Lee, and H.-W. Shen. An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1216 –1224, 2010.