SANDIA REPORT

SAND2013-3435
Unlimited Release
Printed April 2013

Parallel Auto-Correlative Statistics with VTK

Philippe Pébay and Janine Bennett

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2013-3435
Unlimited Release
Printed April 2013

Parallel Auto-Correlative Statistics with VTK

Philippe Pébay
Kitware
26, rue Louis Guérin
69100 Villeurbanne, France
philippe.pebay@kitware.com

Janine Bennett
Sandia National Laboratories
M.S. 9152, P.O. Box 969
Livermore, CA 94551, U.S.A.
jcbenne@sandia.gov

Abstract

This report summarizes existing statistical engines in VTK and presents both the serial and parallel auto-correlative statistics engines. It is a sequel to [PT08, BPRT09b, PT09, BPT09, PT10] which studied the parallel descriptive, correlative, multi-correlative, principal component analysis, contingency, k-means, and order statistics engines. The ease of use of the new parallel auto-correlative statistics engine is illustrated by the means of C++ code snippets and algorithm verification is provided. This report justifies the design of the statistics engines with parallel scalability in mind, and provides scalability and speed-up analysis results for the auto-correlative statistics engine.

Acknowledgments

The authors would like to thank:

- Sandy Landsberg, DOE/ASCR/MAPD Program Manager, for having supported this work,
- Hemanth Kolla, for valuable technical discussions on the methodology and relevance of auto-correlation statistics for post-processing of flame simulations,
- David Thompson, for his earlier contributions to the statistics module of VTK.

Contents

This page intentionally left blank

1 Introduction

This report is a sequel to [PT08, BPRT09b, PT09, BPT09, PT10], which respectively focused on the parallel descriptive, correlative, multi-correlative, principal component analysis, contingency, k-means, and order engines which we designed and implemented as VTK parallel filters; please refer to these references for a detailed presentation of these engines as well as an assessment of their scalability and speed-up properties.

1.1 Initial Motivation: The Titan Informatics Toolkit

The addition of a parallel, scalable statistics module to VTK was motivated by the Titan Informatics Toolkit [WBS08], a collaborative effort between Sandia National Laboratories and Kitware. This effort significantly expanded the Visualization ToolKit (VTK) to support the ingestion, processing, and display of informatics data. By leveraging the VTK data and execution models, Titan provides a flexible, component based, pipeline architecture for the integration and deployment of algorithms in the fields of intelligence, semantic graph and information analysis. A theoretical application

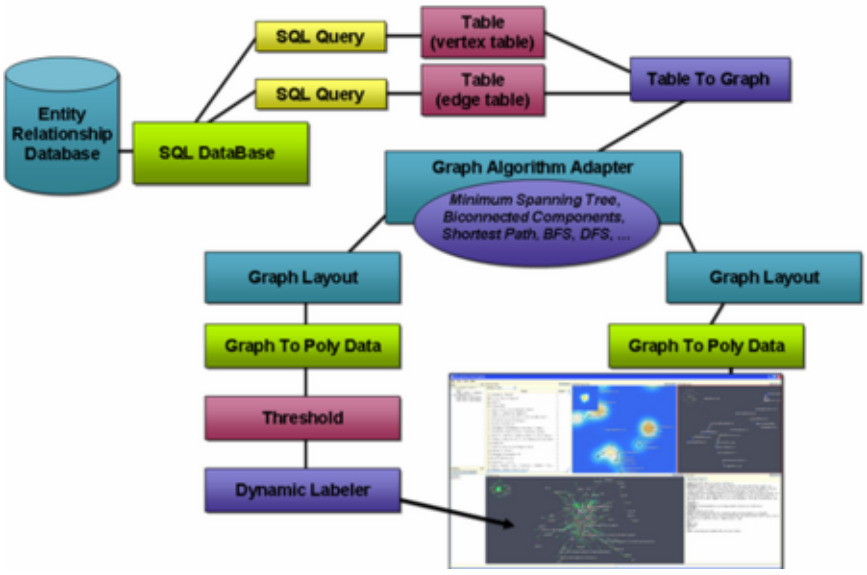

Figure 1. A theoretical application built with Titan.

built from Titan/VTK components is schematized in Figure 1. The flexibility of the pipeline architecture of VTK allows for effective utilization of the Titan components for different problem domains. For instance, an early implementation was OverView, a generalization of the ParaView scientific visualization application dedicated to information visualization, leveraging the ParaView client-server architecture to perform scalable analysis on distributed memory platforms.

In 2008, the parallel statistical engines were integrated into VTK, and the module has continued

to grow as new engines have been developed in the context of the "Network Grand Challenge" LDRD project at Sandia, and under a 3-year grant from the DOE/ASCR program to conduct broad research in the topological and statistical analysis of petascale data. As part of this work, it was discovered that auto-correlative statistics, which were not yet provided by the existing statistics module VTK, where of particular interest to our colleagues in the CRF, due to its use in measuring tubulence spectrum and length scales.

This report thus presents the new parallel, scalable auto-correlative statistics engine which we developed in this context.

1.2 Algorithmic Details: Autocorrelation statistics

Since this report focuses on the *parallelization* of auto-correlative statistics, the following is a brief summary of the definition of such statistical analyses and of their relevance to computational physics, in this case post-processing of Direct Numerical Simulation (DNS) of reactive flows, see Figure 2 for an example flame from a recent DNS run.

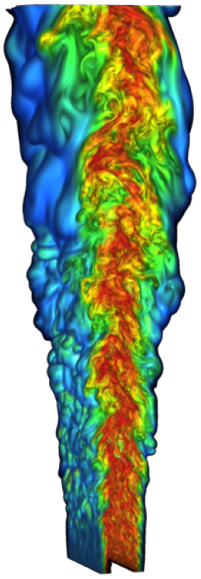

Figure 2. A lifted ethylene jet flame generated from a direct numerical simulation (image courtesy of Hongfeng Yu and Jacqueline Chen).

Auto-correlation is the cross-correlation of a signal with itself, providing a measure of the similarity between observations as a function of the time separation between them. It is typically used to discovery underlying repeating patterns in the presence of noise. Within the context of the analysis of DNS computations, it is the prevalent method used by the combustion community to measure the turbulence spectrum and the turbulence length scales. In an Eulerian framework, the turbulent

eddies in a homogeneous turbulent flow can be perceived as being transported across a point of observation at a rate equal to the mean velocity. The integral time scale, which is computed as the time integration of the autocorrelation function, can be used to deduce the integral length scale through a space-time transformation. The Fourier transform of the autocorrelation function yields the turbulent frequency spectrum, whose integral is the mean square of the velocity fluctuation.

1.3 Statistics Functionality in VTK

A number of univariate, bivariate, and multivariate statistical tools have been implemented in VTK. Each tool acts upon data stored in one or more tables; the first table serves as observations and further tables serve as model data. Each row of the first table is an observation, while the form of further tables depends on the type of statistical analysis. Each column of the first table is a variable.

Operations

In order to meet the two overlapping but not exactly congruent design requirements of matching typical data analysis workflows and being conducive to scalable parallel implementation, our design partitions the statistical analysis algorithms into 4 disjoint operations: learn a model from observations; derive statistics from a model; assess observations with a model, and test a hypothesis. These operations, when all are executed, occur in order as shown in Figure 3. However, it is

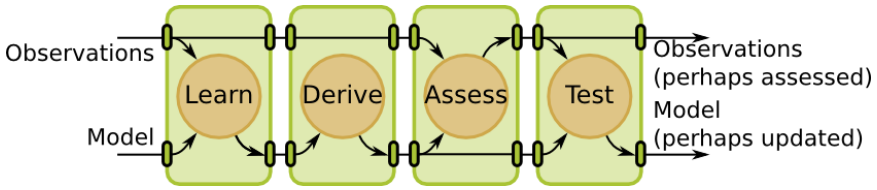

Figure 3. The 4 operations of statistical analysis and their interactions with input observations and models. When an operation is not requested, it is eliminated by connecting input to output ports.

also possible to execute only a subset of these, for example when it is desired that previously computed models, or models constructed with expert knowledge, be used in conjunction with existing data. Note that in earlier publications (e.g., [BPRT09a, PTB10, Inc10]) only the first 3 operations were mentioned; the Test operation, which we initially saw as a part of Derive, was separated out for reasons we explained in [PTBM11]. These operations, performed on a request comprising a set of columns of the input observations table, are further explained as follows:

Learn: Calculate a "raw" statistical model from an input data set. By "raw", we mean the minimal representation of the desired model, that contains only primary statistics. For example, in the case of descriptive statistics: sample size, minimum, maximum, mean, and centered M_2,

M_3 and M_4 aggregates (*cf.* [PÓ8]). For Table 1 with a request $R_1 = \{B\}$, these values are 6, 1, 11, $4.8\bar{3}$, $68.8\bar{3}$, $159.\bar{4}$, and $1759.819\bar{4}$, respectively.

Derive: Calculate a "full" statistical model from a raw model. By "full", we mean the complete representation of the desired model, that contains both primary and derived statistics. For example, in the case of descriptive statistics, the following derived statistics are calculated from the raw model: unbiased variance estimator, standard deviation, and two estimators (g and G) for both skewness and kurtosis. For Table 1 with a request $R_1 = \{B\}$, these additional values are $13.7\bar{6}$, 3.7103, 0.520253, 0.936456, -1.4524, and -1.73616 respectively.

Assess: Given a statistical model – from the same or another data set – mark each datum of a given data set. For example, in the case of descriptive statistics, each datum is marked with its relative deviation with respect to the model mean and standard deviation (this amounts to the one-dimensional Mahalanobis distance). Table 1 shows this distance for $R_1 = \{B\}$ in column E.

Variables

A univariate statistics algorithm only uses information from a single column and, similarly, a bivariate from 2 columns. Because an input table may have many more columns than an algorithm can make use of, the API must provide a way for users to denote columns of interest. Because it may be more efficient to perform multiple analyses of the same type on different sets of columns at once as opposed to one after another, the VTK statistical engines provide a way for users to make multiple analysis requests of a single filter.

Table 1. A table of observations that might serve as input to a statistics algorithm.

row	A	B	C	D	E
1	0	1	0	1	1.03315
2	1	2	2	2	0.76363
3	0	3	4	6	0.49411
4	1	5	6	24	0.04492
5	0	7	8	120	0.58395
6	1	11	10	720	1.66202

As an example, consider Table 1. It has 6 observations of 5 variables. If univariate statistics of A, B, and C are desired then three univariate requests must be made, one for each column. On the other hand, if a multi-variate statistical analysis, such as PCA, is desired $\{A,B,C\}$ then a single request is sufficient.

Algorithms

At the time of writing, the following algorithms are available in VTK:

1. Univariate statistics:
   (a) Descriptive statistics:
      Learn: calculate minimum, maximum, mean, and centered M_2, M_3 and M_4 aggregates;
      Derive: calculate unbiased variance estimator, standard deviation, skewness (1_2 and G_1 estimators), kurtosis (g_2 and G_2 estimators);
      Assess: mark with relative deviations (one-dimensional Mahalanobis distance).
   (b) Order statistics:
      Learn: calculate histogram;
      Derive: calculate arbitrary quantiles, such as "5-point" statistics (quantiles) for box plots, deciles, percentiles, etc.;
      Assess: mark with quantile index.
   (c) Auto-correlative statistics:
      Learn: calculate minimum, maximum, mean, and centered M_2 aggregates for a variable with respect to itself for a set of specified time lags (i.e., time steps between data sets of equal cardinality assumed to represent the same variable distributed in space);
      Derive: calculate unbiased auto-covariance matrix estimator and its determinant, Pearson auto-correlation coefficient, linear regressions (both ways), and fast Fourier transform of the auto-correlation function, again for a set of specified time lags;
      Assess: mark with squared two-dimensional Mahalanobis distance.
2. Bivariate statistics:
   (a) Correlative statistics:
      Learn: calculate minima, maxima, means, and centered M_2 aggregates;
      Derive: calculate unbiased variance and covariance estimators, Pearson correlation coefficient, and linear regressions (both ways);
      Assess: mark with squared two-dimensional Mahalanobis distance.
   (b) Contingency statistics:
      Learn: calculate contingency table;
      Derive: calculate joint, conditional, and marginal probabilities, as well as information entropies;
      Assess: mark with joint and conditional PDF values, as well as pointwise mutual informations.

3. Multivariate statistics: These filters all accept requests containing n_i variables upon which simultaneous statistics should be computed.

   (a) Multi-Correlative statistics:

   Learn: calculate means and pairwise centered M_2 aggregates;

   Derive: calculate the upper triangular portion of the symmetric $n_i \times n_i$ covariance matrix and its (lower) Cholesky decomposition;

   Assess: mark with squared multi-dimensional Mahalanobis distance.

   (b) PCA statistics:

   Learn: identical to the multi-correlative filter;

   Derive: everything the multi-correlative filter provides, plus the n_i eigenvalues and eigenvectors of the covariance matrix;

   Assess: perform a change of basis to the principal components (eigenvectors), optionally projecting to the first m_i components, where $m_i \leq n_i$ is either some user-specified value or is determined by the fraction of maximal eigenvalues whose sum is above a user-specified threshold. This results in m_i additional columns of data for each request R_i.

   (c) k-means statistics:

   Learn: compute optimized set(s) of cluster centers from initial set(s) of cluster centers. In the default case, the initial set comprises the first k observations. However, the user can specify one or more sets of cluster centers (with possibly differing numbers of clusters in each set) via an optional input table, in which case an optimized set of cluster centers is computed for each of the input sets.

   Derive: calculate the global and local rankings amongst the sets of clusters computed in the learn operation. The global ranking is the determined by the error amongst all new cluster centers, while the local rankings are computed amongst clusters sets with the same number of clusters. The total error is also reported;

   Assess: mark wth closest cluster id and associated distance for each set of cluster centers.

A utilization example of the statistical engines of VTK in ParaView is shown in Figure 4. Specifically, a PCA analysis is performed on a quadruple of variables of interest in a 2D flame simulation, whereby the statistical model is calculated (learn and derive operations) on a randomly-sampled subset of $\frac{1}{10}$-th of the entire data set, after which all points in the data set are marked with their respective relation deviations from this model.

1.4 Input and Output Ports

The statistics algorithms have by default 3 input ports and 3 output ports as follows:

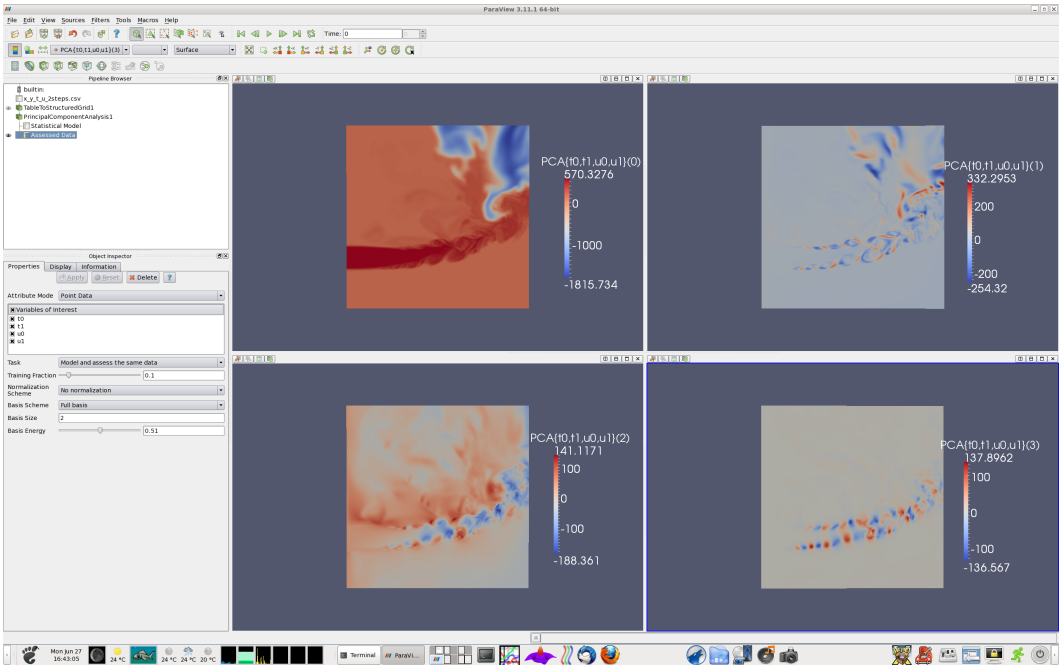

Figure 4. Several of the VTK parallel stastistics engines are integrated into ParaView. Example using PCA on 4 data set attributes.

Input Port 0: This port is identified as `vtkStatisticsAlgorithm::INPUT_DATA` and is used for learn data.

Input Port 1: This port is identified as `vtkStatisticsAlgorithm::LEARN_PARAMETERS` and is used for learn parameters (e.g., initial cluster centers for *k*-means clustering, time lags for auto-correlation).

Input Port 2: This port is identified as `vtkStatisticsAlgorithm::INPUT_MODEL` and is used for a priori models.

Output Port 0: This port is identified as `vtkStatisticsAlgorithm::OUTPUT_DATA` and mirrors the input data, plus optional assessment columns.

Output Port 1: This port is identified as `vtkStatisticsAlgorithm::OUTPUT_MODEL` and containts any generated model.

Output Port 2: This port is identified as `vtkStatisticsAlgorithm::OUTPUT_TEST` and is currently experimental and not used by all statistics algorithms[1].

[1] In earlier implementations and reports it was called `vtkStatisticsAlgorithm::ASSESSMENT`, a key which has been deprecated since.

All input and output ports are of type `vtkTable`, with the exception of both input and output ports 1 which are of type `vtkMultiBlockDataSet`. Note that in earlier implementations of these filters it was also possible for ports 1 to be of type `vtkTable`, however this is no longer the case.

In the following sections, we present implementation details on the serial and parallel versions of the auto-correlative statistics algorithm, provide a basic user manual thereof, provide verification results, and examine the scalability of the parallel implementation.

2 Parallel Statistics Classes

2.1 Implementation Details

We build a full statistical model in two separate operations, as this enables both database normalization (resulting from the fact that there is no redundancy in the primary model) and parallel computational efficiency. In our approach inter-processor communication and updates are performed only for primary statistics. The calculations to obtain derived statistics from primary statistics are typically fast and simple and need only be calculated once, without communication, upon completion of all parallel updates of primary variables. Data to be assessed is assumed to be distributed in parallel across all processes participating in the computation, thus no communication is required as each process assesses its own resident data.

Therefore, in the parallel versions of the statistical engines, inter-processor communication is required only for the Learn operation, while both Derive and Assess are executed in an embarrassingly parallel fashion due to data parallelism. This design is consistent with the data parallelism methodology used to enable parallelism within VTK, most notably in ParaView. Because the focus of this report is on the parallel speed-up properties of statistics engines, we will not report on the Derive or Assess operations, as these are executed independently from each other, on a separate process for each part of the data partition. However, because the Derive operation provides the derived quantities to which one is naturally accustomed (e.g., variance as opposed to M_2 aggregate), the numerical results reported here are those that are yielded by the consecutive application of the Learn and then Derive operations.

At the time of writing (February 2013), the following 8 parallel statistics engines are implemented and available in the parallel statistics module of VTK.

1. `vtkPAutoCorrelativeStatistics`;
2. `vtkPDescriptiveStatistics`;
3. `vtkPOrderStatistics`;
4. `vtkPCorrelativeStatistics`;
5. `vtkPContingencyStatistics`;
6. `vtkPMultiCorrelativeStatistics`;
7. `vtkPPCAStatistics`;
8. `vtkPKMeansStatistics`.

Each of these parallel algorithms is implemented as a subclass of the respective serial version of the algorithm and contains a `vtkMultiProcessController` to handle inter-processor communication. Within each of the parallel statistics classes, the Learn operation is the only operation

whose behavior is changed (by reimplementing its virtual method or by reimplementing virtual methods that are called by the Learn operation) due to the data parallelism inherent in the Derive and Assess operations. The Learn operation of the parallel algorithms performs two primary tasks:

1. Calculate statistics on local data by executing the Learn code of the superclass.
2. If parallel updates are needed (i.e. the number of processes is greater than 1), perform necessary data gathering and aggregation of local statistics into global statistics.

The descriptive, correlative, auto-correlative and multi-correlative statistics algorithms perform the aggregation necessary for the statistics which they are computing using the arbitrary-order update and covariance update formulas presented in [PÓ8]. Because the PCA statistics class derives from the multi-correlative statistics algorithm and inherits its Learn operation, a static method is defined within the parallel multi-correlative statistics algorithm to gather all necessary statistics, cf. [BPRT09b] for details. As explained in the aforementioned references, all of these parallel classes exhibit optimal parallel speed-up properties. Similarly, the contingency statistics class derives from the bivariate statistics class and implements its own aggregation mechanism for the Learn operation. However, unlike the other statistics algorithms which rely on statistical moments (descriptive, correlative, auto-correlative, multi-correlative, PCA, and k-means), this aggregation operation is, in general, not embarrassingly parallel and, therefore, optimal parallel scale-up is not observed when this class is used outside of its intended domain of applicability, as explained in [PT09]. The same is the case for the order statistics, since the parallel update of a histogram involves in general more than negligibly small amounts of data as compared to the overall data set size.

2.2 Usage

It is fairly easy to use the serial statistics classes of VTK; it is not much harder to use their parallel versions. All that is required is a parallel build of VTK and a version of MPI installed on your system.

For example, Listing 1 demonstrates how to calculate auto-correlative statistics, in parallel, on each column of an input set `inData` of type `vtkTable*` with an associated set of input parameters and no subsequent data assessment. It is assumed that this input data type be of numeric type (i.e., double), with the following storage convention: each time-step corresponds to a block of data of the same size, denoted `nVals` above. Each such block is often referred to as a *time-slab*. As a result, there are as many data points for each variable as there are time steps, times the slab size `nVals`. Last, a parameter table `paramTable` contains the list of *time lags* of interest, i.e., those time steps for which the auto-correlation must be computed with respect to the initial time step (the first slab in the data set). In particular, if this parameter table contains only one entry with value 0, then the auto-correlation of the entire dataset against itself will be calculated, which will lead to a covariance matrix with all coefficients equal to the variance of the variable, and the auto-correlation coefficient will be equal to 1.

For univariate statistics algorithms, calling `AddColumn()` for each column of interest is sufficient – each request R_i can by definition only reference a single column and so the filter automatically turns each column of interest into a separate request. However, this is not sufficient for multivariate filters as each request might have a different number of columns of interest. In this case, requests for columns of interest are specified by calling `SetColumnStatus()` multiple times to identify the variables to be used, followed by a call to `RequestSelectedColumns()`. For more details on the use of multivariate filters, please refer to, e.g., [BPT09].

The examples thus far assume that a MPI communicator was previously prepared within the parallel environment used to execute the parallel auto-correlative statistics engine. It is outside the scope of this report to discuss I/O issues, and in particular how a `vtkTable` can be created and filled with the values of the variables of interest. See VTK's online documentation for details [vtk]. However, we include a small amount of code to prepare a parallel controller.

In the code example from Listing 1, the `vtkMultiProcessController` object passed to `Foo()` is used to determine the set of processes (which may be a subset of a larger job) among which input data is distributed. VTK uses subroutines of this form to execute code across many processes. In Listing 2 we demonstrate that, to prepare a parallel controller to execute `Foo()` in parallel using MPI, one must first (e.g. in the main routine) create a `vtkMPIController` and pass it the address of `Foo()`. Note that, when using MPI, the number of processes is determined by the external program which launches the application.

```
void Foo( vtkMultiProcessController* controller, void* arg )
{
  // Use the specified controller on all parallel filters by default:
  vtkMultiProcessController::SetGlobalController( controller );

  // Assume the input dataset is passed to us:
  vtkTable* inputData = static_cast<vtkTable*>( arg );

  // Create parallel auto-correlative statistics class
  vtkPAutoCorrelativeStatistics* pas = vtkPAutoCorrelativeStatistics::New();

  // Set input data port
  pas->SetInput( 0, inputData );

  // Select all columns in inputData
  for ( int c = 0; c < inputData->GetNumberOfColumns(); ++ c )
    {
    pas->AddColumn( inputData->GetColumnName[c] );
    }

  // Set spatial cardinality
  pas->SetSliceCardinality( nVals );

  // Set parameters for autocorrelation of whole data set with respect to itself
  pas->SetInputData( vtkStatisticsAlgorithm::LEARN_PARAMETERS, paramTable );

  // Calculate statistics with Learn and Derive operations only
  pas->SetLearnOption( true );
  pas->SetDeriveOption( true );
  pas->SetAssessOption( false );
  pas->SetTest( false );
  pas->Update();
}
```

Listing 1: A subroutine – that should be run in parallel – for calculating auto-correlative statistics.

```
vtkTable* inputData;
vtkMPIController* controller = vtkMPIController::New();
controller->Initialize( &argc, &argv );

// Execute the function named Foo on all processes
controller->SetSingleMethod( Foo, &inputData );
controller->SingleMethodExecute();

// Clean up
controller->Finalize();
controller->Delete();
```

Listing 2: A snippet of code demonstrating how to execute a subroutine (`Foo()`) in parallel. In reality, `inData` would be prepared in parallel by `Foo()` but is assumed to be pre-populated here to simplify the example.

3 Results

We performed a series of tests on `Titan`, the primary system of the National Center for Computational Sciences at Oak Ridge National Laboratory. The Cray XK7 system contains 18,688 nodes connected by a Gemini interconnect, with each holding a 16-core AMD Opteron 6274 processor and 32 GB of memory.

3.1 Algorithm Scalability

In order to assess speed-up independently of the load-balancing scheme, a series of (pseudo-) randomly-generated samples is used. Specifically, input tables are created at run time by generating 4 separate samples of independent pseudo-random variables, the two first (resp. last) variables having a standard normal (resp. standard uniform) distribution. Since our objective is to assess the scalability of the parallel statistics engines only, equally-sized slabs of data are created by each process in order to work with perfectly load-balanced cases. For the same reason, the amount of time needed to create the input data table is excluded from the analysis. In this test, `vtkPAutoCorrelativeStatistics`, with Learn, Derive, and Assess modes on, is executed for each of the 4 columns, and the corresponding wall clock time is reported.

With this synthetic test case, we assess:

1. relative speed-up (at constant total work), and
2. scalability of the rate of computation (at constant work per processor).

Strong Scaling

Given a problem of size N (as measured in our case by sample size), the wall clock time measured to complete the work with p processors is denoted $T_N(p)$. Then, strong scaling with p processors is defined as the following ratio:

$$S_N(p) = \frac{T_N(1)}{T_N(p)}.$$

Optimal (linear) speedup is attained with p processors when $S_N(p) = p$ and, therefore, strong scaling results for S_N may be visually inspected by plotting S_N *versus* the number of processors: optimal speed-up is revealed by a line, the angle bisector of the first quadrant.

In order to assess strong scaling, we use a test case that with a pseudo-random samples of total size $N = 1,562,624$ per time step, with increasing number of processors $p = 2^k$ with $k \in \{6,\ldots,15\}$. As the smallest number of processors is 64 and not 1, for convenience and legibility of the results we normalize the formula for $S_N(p)$ with a multiplicative factor of 64: this allows us to keep the handy visual comparison versus the first bisector.

Table 2. Strong scaling (at constant total work), with a total sample size of $N = 1,562,624$ doubles per time step.

N/p	p	Auto-correlative (sec. / $S_N(p)$)
1,562,624	64	18.1521 / 64
781,312	128	9.23122 / 125.85
390,656	256	4.83677 / 240.19
195,328	512	2.62850 / 441.98
97,664	1024	1.53182 / 758.40
48,832	2048	0.973303 / 1193.6
24,416	4096	0.750859 / 1547.2
12,208	8192	0.581591 / 1997.5
6,104	16384	0.580288 / 2002.0
3,052	32768	0.598636 / 1940.6

The wall clock times obtained on `titan` are provided in Table 2 and the corresponding strong scaling numbers are plotted in Figure 5. As expected based on the embarrassingly parallel nature of the Learn-Derive-Assess pattern for moment-based statistics, which we explained in detail in particular in [BPRT09a], we observe that the measured strong scaling numbers are optimal until the decreasing amount of work per process ultimately results in a situation where overheads, even small in absolute terms, become dominant as compared to the amount of actual computational work. In this current example, it appears that with 1024 processes, i.e., with a per-process of load below 10^5 points per time step, strong scaling begins to tail off. Furthermore, with less than 10^4 points per time step and per process, noticeable speed down begins to appear, as expected with such a small load per process where operating sytem overheads become dominant.

Weak Scaling

Weak scaling is defined as the following ratio:

$$r(p) = \frac{N(p)}{T_{N(p)}(p)},$$

where $N(p)$, the sample size, now varies with the number of processors p. We then measure its scalability by normalizing it with respect to the rate of computation obtained with a single processor, as follows:

$$R(p) = \frac{r(p)}{r(1)} = \frac{N(p)T_{N(1)}(1)}{N(1)T_{N(p)}(p)},$$

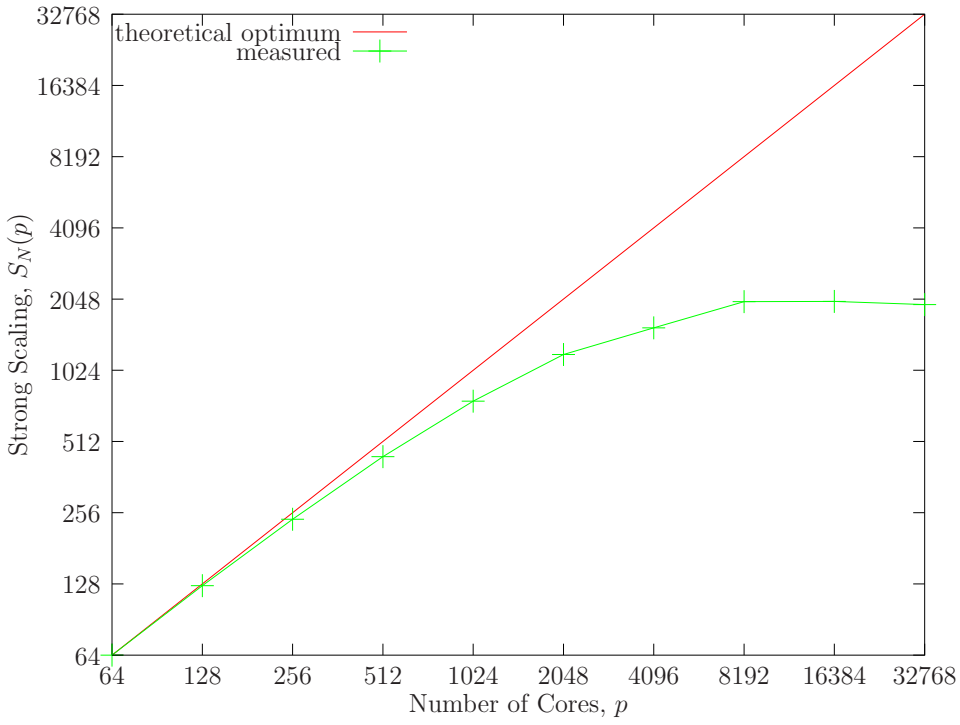

Figure 5. Strong scaling at constant total work with a total data size of $N = 1,562,624$ doubles per time step.

In particular, if the sample size is made to vary in proportion to the number of processors, i.e., if $N(p) = pN(1)$, then

$$R(p) = \frac{pT_{N(1)}(1)}{T_{pN(1)}(p)} = \frac{pT_{N(1)}(1)}{pT_{N(1)}(p)} = \frac{T_{N(1)}(1)}{T_{N(1)}(p)},$$

and thus, optimal (linear) scalability is also attained with p processors when $R(p) = p$. Note that without linear dependency between N and p, the latter equality no longer implies optimal scalability. Hence, under the above assumptions, scalability can also be visually inspected, with a plot of R *versus* the number of processors, where optimal scalability is also indicated by the angle bisector of the first quadrant.

Weak scaling is now assessed using the same test case as in § 3.1, with the difference that, in order to maintain a constant work per processor, increasingly large samples are created: specifically, each data sets contains $N(p) = np$ doubles per time step, where $n = 10^5$ and $p = 2^k$ with $k \in \{6, \ldots, 15\}$ respectively denote the number of sample points per processor and the number of processes.

The wall clock times measured on `titan` are given in Table 3 and the corresponding weak scaling numbers are plotted in Figure 6. These exhibit near-optimal scalability, with an overall order above 0.98. These findings confirm what we explained theoretically for all parallel, moment-based statistics algorithms using the Learn-Derive-Assess pattern, and confirm experimentally for concrete types of such algorithms, in particular in [BPRT09a].

Table 3. Weak scaling (at constant load per processor) with a per processor load of $N(p) = p \times 10^5$ doubles per time step.

$N(p)$ ($\times 10^5$)	p	Auto-correlative (sec. / R)
64	64	1.49354 / 64
128	128	1.48828 / 128.52
256	256	1.50946 / 253.30
512	512	1.55391 / 492.11
1024	1024	1.53471 / 996.53
2048	2048	1.57202 / 1945.8
4096	4096	1.61576 / 3786.2
8192	8192	1.60698 / 7613.7
16384	16384	1.66449 / 14701
32768	32768	1.67711 / 29181

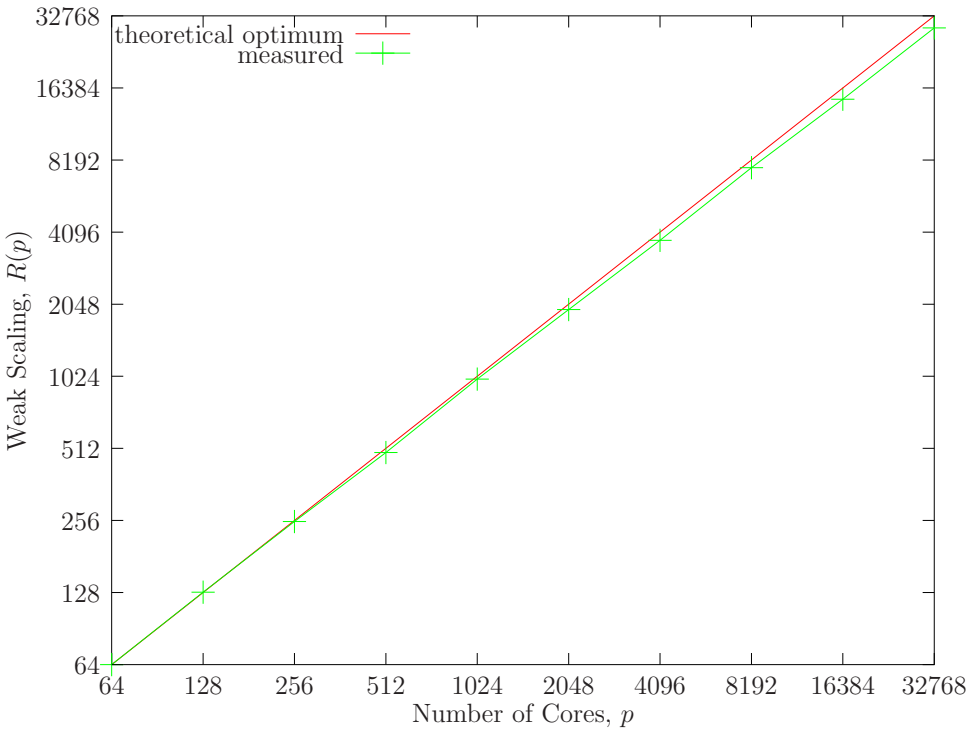

Figure 6. Weak scaling at constant work per processor of $N(p)/p = 10^5$.

3.2 Algorithm Correctness

In order to assess the algorithm correctness of `vtkPAutoCorrelativeStatistics`, we have examined the statistical models obtained when both Learn and Derive options are turned on with a variety of input data sets, both very small (which could then be verified point by point) and very large.

In this report, we do not report on each of these test cases, which would present very little interest, other than saying that for those who were too large to for manual verification, the validation method verifies that:

1. when the time lag is chosen to be 0, and the spatial cardinality is equal to the total data set cardinality, then the calculated covariance matrix has all coefficients equal to the variance calculated over the entire data set with `vtkPCorrelativeStatistics`, which has already been validated independently [PT08].

2. when the time lag is chosen to be greater than 0, and the input tables are generated at run time by pseudo-random, independent sampling, then the Pearson auto-correlation coefficient must be equal to 0 within a very small numerical tolerance.

References

[BPRT09a] J. Bennett, P. Pébay, D. Roe, and D. Thompson. Numerically stable, single-pass, parallel statistics algorithms. In *Proc. 2009 IEEE International Conference on Cluster Computing*, New Orleans, LA, August 2009.

[BPRT09b] J. Bennett, P. Pébay, D. Roe, and D. Thompson. Scalable multi-correlative statistics and principal component analysis with Titan. Sandia Report SAND2009-1687, Sandia National Laboratories, March 2009.

[BPT09] J. Bennett, P. Pébay, and D. Thompson. Scalable *k*-means statistics with Titan. Sandia Report SAND2009-7855, Sandia National Laboratories, November 2009.

[Inc10] Kitware Inc. *The VTK User's Guide, version 5.4*. Kitware, Inc., 2010.

[PÓ8] P. Pébay. Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. Sandia Report SAND2008-6212, Sandia National Laboratories, September 2008.

[PT08] P. Pébay and D. Thompson. Scalable descriptive and correlative statistics with Titan. Sandia Report SAND2008-8260, Sandia National Laboratories, December 2008.

[PT09] P. Pébay and D. Thompson. Parallel contingency statistics with Titan. Sandia Report SAND2009-6006, Sandia National Laboratories, September 2009.

[PT10] P. Pébay and D. Thompson. Parallel order statistics with Titan. Sandia Report SAND2010-6466, Sandia National Laboratories, September 2010.

[PTB10] P. Pébay, D. Thompson, and J. Bennett. Computing contingency statistics in parallel: Design trade-offs and limiting cases. In *Proc. 2010 IEEE International Conference on Cluster Computing*, Heraklion, Crete, Greece, September 2010.

[PTBM11] P. Pébay, D. Thompson, J. Bennett, and A. Mascarenhas. Design and performance of a scalable, parallel statistics toolkit. In *Proc. 25th IEEE International Parallel & Distributed Processing Symposium, 12th International Workshop on Parallel and Distributed Scientific and Engineering Computing*, Anchorage, AK, U.S.A., May 2011.

[vtk] VTK Doxygen documentation. http://www.vtk.org/doc/nightly/html.

[WBS08] B. Wylie, J. Baumes, and T. Shead. Titan informatics toolkit. In *IEEE Visualization Tutorial*, Columbus, OH, October 2008.

DISTRIBUTION:

2 Philippe Pébay, Kitware
26 avenue Louis Guérin, 69100 Villeurbanne cedex, France

2 MS 9152 Janine Bennett, 8953

1 MS 9152 Robert Clay, 8953

1 MS 9159 Jim Costa, 8953

2 MS 9018 Central Technical Files, 8944

1 MS 0899 Technical Library, 9536

Sandia National Laboratories