





Approaches for In Situ Computation of Moments in a Data-Parallel Environment

Karen C. Tsai^{†1} , Roxana Bujack^{‡1} , Berk Geveci^{§2}, Utkarsh Ayachit^{¶2}  and James Ahrens^{||1} 

¹Los Alamos National Laboratory

²Kitware, Inc.

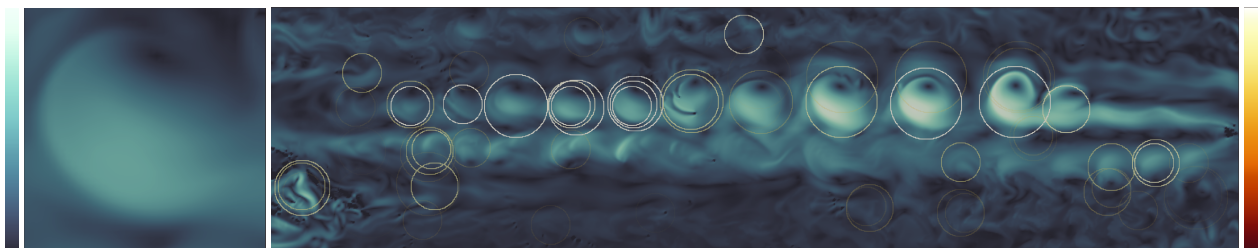


Figure 1: Example pattern (left) and the corresponding pattern detection result using the moment invariants algorithm in a selected subset of the MPAS ocean dataset (right). The matches are represented by circles, whose diameter correspond to the size for which the match was found and color to the similarity to the pattern with the colormap shown on the right. The pattern and the field are color coded by the magnitude of the velocity vector field with the colormap shown on the left.

Abstract

Feature-driven in situ data reduction can overcome the I/O bottleneck that large simulations face in modern supercomputer architectures in a semantically meaningful way. In this work, we make use of pattern detection as a black box detector of arbitrary feature templates of interest. In particular, we use moment invariants because they allow pattern detection independent of the specific orientation of a feature. We provide two open source implementations of a rotation invariant pattern detection algorithm for high performance computing (HPC) clusters with a distributed memory environment. The first one is a straightforward integration approach. The second one makes use of the Fourier transform and the Cross-Correlation Theorem. In this paper, we will compare the two approaches with respect to performance and flexibility and showcase results of the in situ integration with real world simulation code.

CCS Concepts

• **Human-centered computing** → Visualization; • **Theory of computation** → Parallel algorithms; Pattern matching;

1. Introduction

In modern supercomputer architectures, the computational capacity is generally larger than the I/O bandwidth [CPA*10]. The goal in an in situ approach is to perform data processing and analysis during the generation of the raw data in order to reduce output

data. Scientists running large simulations need to decide in situ which parts of their data they want to store for post-hoc analysis, prioritizing saving the more important regions of the data. Wherever the data has scientifically meaningful features or patterns, in situ algorithms must find and save those patterns.

The challenges thereby are that suitable algorithms must be able to decide what is worth keeping during the run of the simulation in an autonomous way and they must further be compatible with the simulation in three ways. First, they must be able to be integrated with the simulation's code. Second, they must be able to run in the same hardware settings and data distribution that the simulation uses. Third, they may not slow down the simulation run significantly.

[†] e-mail: ktsai@lanl.gov

[‡] e-mail: bujack@lanl.gov

[§] e-mail: berk.geveci@kitware.com

[¶] e-mail: utkarsh.ayachit@kitware.com

^{||} e-mail: ahrens@lanl.gov

The motivation is the need of a scalable algorithm that can detect patterns of interest during the runtime of large simulations using data distribution to achieve data reduction and enable in situ visualization without sacrificing the accuracy in regions of interest.

For this paper, we assume that the application scientist has *a priori* information on patterns within the data that are of interest, e.g. Figure 1. Therefore, we want to run an in situ pattern detection algorithm to focus on areas with the patterns of interest. Then we can either provide an in situ occlusion-free visualization of the region [BRA18] or reduce the data by storing these regions of interest in a sufficiently high resolution for post-processing, while storing other less interesting areas in a coarser resolution.

Pattern detection algorithms often have to look at every possible orientation of the given template in order to determine if there is a match, potentially causing a significant load on performance. This determination is not practical for an in situ environment. In developing the in situ pattern detection algorithm, we use orientation invariant characteristics to avoid performance load resulting from checking every possible angle. These characteristics are called moment invariants and they are used for determining the degree of similarity between the given pattern template and the data in our pattern detection algorithm [FSZ16, BH17].

Moments are projections of a function—in our case the pattern and the field—to a function space basis. The similarity between pattern and field corresponds to the reciprocal of the Euclidean distance of their coordinates in the function space, that is, the moments. The moments can be transformed into moment invariants using the right transformation. This process is called normalization and can be imagined as projecting the function space onto a lower dimension in the way that all instances of a function that only differ by a rotation are projected onto the same point. The Euclidean distance in this reduced space then anti-correlates with the similarity independent of the orientation difference between the pattern and the field.

In this paper, we analyze two different approaches that we implemented for the in situ data-parallel computation of moment invariants: straightforward integration and fast Fourier transform (FFT) both in C/C++. If the data were not distributed, the FFT would have an advantage as the data size increases. But in our setting, this process is not obvious because it requires global communication, while our integration-based implementation only needs communication across nodes with data that is locally close. Especially in cutting edge HPC architectures, the communication costs have a great influence. Our experiments investigate the resulting computation vs communication speed trade-off.

All algorithms are publicly available through the modules `MomentInvariants` and `ParallelMomentInvariants` [Kit] of the Visualization Toolkit (VTK) [SLM04].

The main contributions of this work include the following:

- First presentation of an in situ, data-parallel moment invariants algorithm.
- Comparison of speed between a direct approach with local communication, and an FFT approach with global communication w.r.t. parameter variations and weak and strong scaling.
- Open source release of the complete module as part of the Visualization Toolkit (VTK).

- Demonstration of in situ integration with MFX Exa [MEb, MEa] and WarpX [VAB*18] simulations of the Exascale Computing Project.

2. Related Work

Moment Invariants were introduced to the image processing community by Hu [Hu62] in 1962. He suggested a set of seven invariants with respect to translation, rotation, and scaling for two-dimensional scalar images. Dirilten and Newman were the first to use the moment tensors. In their seminal work [DN77], they showed that the moment tensor contractions to zeroth order are invariant under orthogonal transformations. Sadjardi and Hall [SH80] provided three moment invariants for three-dimensional scalar fields. Pinjo, Cyganski, and Orr [PCO85] calculated 3D orientation estimation from moment contraction to first-order moments. Flusser [Flu00] presented the first calculation rule for a complete and independent 2D scalar moment basis. For the tensor contraction method, as described by Dirilten and Newman [DN77], it is hard to judge completeness and independence of the resulting set of invariants. Suk and Flusser calculated a complete set and afterwards skipped the linearly dependent ones in [SF11]. Higher-order dependencies still remain.

Langbein and Hagen [LH09, Lan14] showed that tensor contraction method can be generalized to arbitrary tensor fields. Schlemmer et al. [SHM*07] were the first to introduce a set of five 2D moment invariants for vector fields and they utilized the FFT for acceleration. Bujack et al. [BHSH15] present an algorithm that provides a complete and independent set of two-dimensional moment invariants for vector fields and later an algorithm for invariants of 3D vector fields [BKH*15]. For a comprehensive introduction to moment invariants, we refer the interested reader to [FSZ16]. In our implementation, we will use the general algorithm presented in [BH17]. This algorithm can compute moment invariants for two-dimensional as well as three-dimensional scalar, vector, and tensor fields.

There have been several approaches to accelerate the computation of the moments of scalar data for image comparison tasks. They include image decomposition [SHF12], Green's theorem [YA96], image slicing [PKK08, SB11], and recursive algorithms [Che90]. For some orthonormal moment bases, acceleration can be achieved through recurrence relations [Kin76, MR95, HSS11]. In this application, the moments need to be computed only once in the center of the template and the image, and no speedup can be gained using an FFT approach. For pattern detection, a great acceleration option lies in the convolution theorem and the FFT [SHM*07].

We will extend the existing work by suggesting a parallel algorithm that is able to run in data distributed setting and in situ, which will enable these algorithms to be used with large scale simulations on high performance computers.

In situ visualization is a growing field of interest [RCMS12, BAA*16]. Two main approaches of integration can be distinguished, first direct embedding of the routines as part of the simulation, e.g. [YWG*10, WPS*15, DCH*16], which cannot be reused, and second general-purpose libraries that can be included into different simulations, like ADIOS [LZKS09], ParaView Catalyst [ABG*15], Ascent [Asc], Visit libSIM [lib, KPZ11], SENSEI [AWW*16]. We will follow both approaches in this work.

3. Theory

Moments are the projections of a function with respect to a function space basis. We can think of them as the coordinates that represent the pattern. They can then be used to construct moment invariants—values that do not change under certain transformations. We will follow the normalization approach for the construction of moment invariants. This approach means a standard position is defined by demanding certain moments to assume fixed values and all functions are transformed to match it. Then, the remaining moments form a complete and independent set of moment invariants.

Dirilten and Newman suggest the use of moment tensors for the construction of moment invariants through tensor contraction for scalar functions in [DN77]. Langbein et al. [LH09] generalized the definition of the moment tensor to tensor valued functions.

Definition 1 For a tensor field $T : \mathbb{R}^d \rightarrow \mathbb{R}^{d^n \times d^m}$ with compact support, the **moment tensor** oM of order $o \in \mathbb{N}$ takes the shape

$${}^oM = \int_{\mathbb{R}^d} x^{\otimes o} \otimes T(x) d^d x, \quad (1)$$

where $x^{\otimes o}$ denotes the o -th tensor power of the vector x .

Theorem from [BH17] is the foundation of the algorithm.

Theorem 1 The moment tensor of order o of a tensor field of covariant rank m , contravariant rank n , and weight w is a tensor of covariant rank m , contravariant rank $n + o$, and weight $w - 1$.

It follows from tensor algebra that all first-rank contractions behave like vectors under rotation and reflection [PCO85], which allow us to define a standard position of a pattern for tensor fields of arbitrary rank analogously to the PCA in the scalar case. A detailed description of the algorithm and the proof of Theorem 1 can be found in [BH17].

4. Serial Algorithm

The algorithm consists of two main steps. A schematic overview of the workflow and architecture is given in Figure 2. The first step is the computation of the moments in the dataset. For all considered positions in the dataset, we need to approximate Eq. (1) numerically. It is not complicated, but is computationally expensive. Our algorithm supports two different ways to compute the moments.

4.1. Integration

Step one performs a discrete approximation to the direct integration of each moment:

$${}^{(p+q+r)}M_j^{p,q,r,i} = \int_{B_R(0)} x^p y^q z^r T_j^i(x, y, z) dx dy dz, \quad (2)$$

where $p, q, r \in \mathbb{N}$; $p + q + r = o$ is the combination of indices that define one specific basis function of order o ; T_j^i refers to one component of the tensor field T ; and $B_R(0) \subset \mathbb{R}^3$ is the ball around the origin with radius R .

The integration approach has an input parameter *numberOfIntegrationSteps*, which steers the accuracy of the discrete integration. Depending on this parameter, a uniform stencil (a sliding window) is generated over which we evaluate

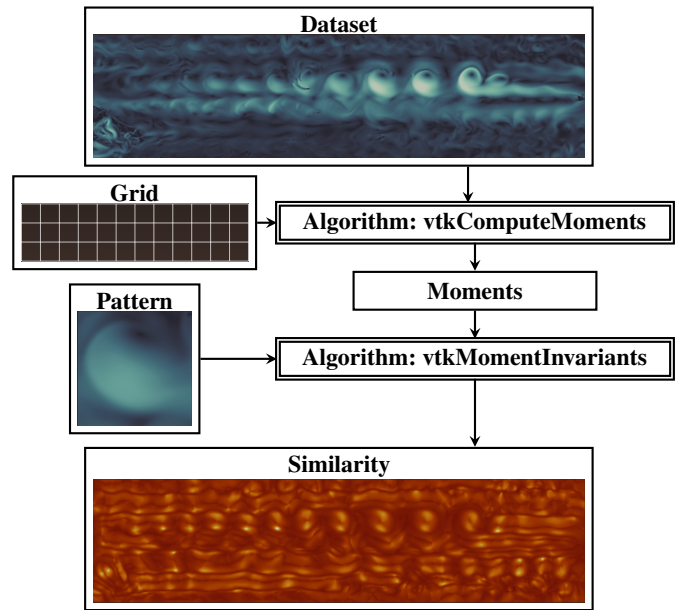


Figure 2: Generation of the spatial areas of interest through the moment invariants module with example images.

the finite sum that approximates the moments. If the user sets *numberOfIntegrationSteps* to zero, the second option is used in which the original grid of the dataset is used to evaluate the integral. Both options takes $O(\tilde{N}M)$ operations, where \tilde{N} is the number of points in the second input dataset “Grid” and M is the number of points that lie in a ball of the given radius.

4.2. FFT

This approach is based on the FFT and the cross-correlation theorem:

$$f \star g = \mathcal{F}^{-1} \{ \overline{\mathcal{F}\{f\}} \cdot \mathcal{F}\{g\} \}, \quad (3)$$

as Eq. (2) applied to all points in the grid can be interpreted as a cross-correlation,

$$\begin{aligned} {}^{(p+q+r)}M_j^{p,q,r,i}(x, y, z) &= \int_{B_R(x,y,z)} (x-x')^p (y-y')^q (z-z')^r \\ &\quad \cdot T_j^i(x', y', z') dx' dy' dz' \\ &= (x^p y^q z^r \star T_j^i)(x, y, z). \end{aligned} \quad (4)$$

This approach takes $O(N \log N)$ operations with N being the number of points in the original input dataset.

The FFT approach can leverage existing FFT libraries to compute the Fourier transform and the inverse Fourier transform. For our serial FFT implementation, we use the *KISSFFT* package, a mixed-radix FFT algorithm in one or more dimensions [Bor]. Before we invoke FFT library function calls, our algorithm carries out a two-step padding on the input dataset and then the corresponding kernel dataset that will offset the cyclic nature of FFT in frequency space to compute the cross-correlation correctly. The algorithm first pads the input dataset around all boundaries with half of the size of the

kernel dataset, which is the radius parameter, and then it pads the dataset to a square. After the input dataset is padded, the algorithm makes sure the corresponding kernel dataset is of the same size by zero-padding. Later, when the algorithm has the output from the inverse Fourier transform on the element-wise multiplications, the output is truncated back to the size of the original input dataset.

4.3. Normalization and Comparison

The second part of the algorithm takes the pattern and the moments of the dataset and obtains moment invariants by normalization as in [BH17]. It computes the moment tensors of the pattern, computes all products, contracts them, and chooses the most robust candidates for the normalization. Then, it normalizes all moment tensors with respect to this choice and computes the rotation invariant similarity between the normalized moments of pattern and each point in the dataset. The similarity value is derived as the inverse Euclidean distance between the moment invariants of the dataset and the pattern. The result is a scalar field with the extent of the dataset that is high if it matches the pattern and low otherwise. This part is more difficult technically to implement, but not very expensive with respect to computation time. Figure 1 shows the input dataset overlaid with the output similarity scalar field that is visualized with circles scaled by the corresponding radius and colored by the similarity values.

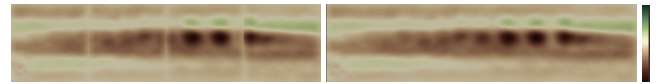
Both parts are incorporated as two separate *VTK* filters. The separation allows the search for many patterns without having to recompute the moments. Since the moments do not depend on the pattern, this avoids a repeat of the computationally expensive part.

5. Parallel Algorithm

The normalization of the moments and their comparison are completely local operations. We have a set of moments on each pixel, one set for the pattern, and do not need any other information. That means even in a data-parallel environment, the filter *vtkMomentInvariants* of the *MomentInvariants* module is able to correctly perform the pattern matching if it is given the correct moments of the data it owns. The parallel computation of the moments on the other hand needs special attention if the integration is performed across node boundaries. The stencil that is used for the numerical integral approximation needs information from the surrounding areas, and this area depends on the scale on which the pattern is searched. As a result, we cannot make use of ghost cells because, for very large radii, the pattern size might exceed the diameter of the data stored in one node, which would lead to a prohibitive number of ghost cells.

Figure 3 shows the result of the parallel algorithm compared to its serial counterpart on the selected subset of an MPAS ocean dataset distributed over four nodes. The serial algorithm is agnostic with respect to its neighbors, and therefore cannot compute the moments near the boundaries. The data-parallel algorithm on the other hand is able to correctly communicate across node boundaries. Both approaches return the correct result, which coincides with the output if the whole data is on one single node.

Here the separation of the algorithm in its two separate filters (*vtkComputeMoments* and *vtkMomentInvariants*) pays off as well because we can reuse the second filter without any changes in the parallel setting and only need to parallelize the first one.



(a) Naive serial implementation (b) Both parallel approaches provide the correct result.

Figure 3: Comparison of the serial and the parallel algorithms to compute moments on the selected subset of an MPAS ocean dataset distributed over four ranks. The image shows the zeroth order moments using an integration radius of 3, which corresponds to 1/10th of the height of the dataset.

We added a module to *VTK* that contains the new filter *vtkPComputeMoments*, which is a subclass of the serial *vtkComputeMoments*, Figure 2. It inherits its basic structure and only re-implements the different options of the computation of the moments. The overall flow of the parallel pattern detection stays the same. The only difference is that we call *vtkPComputeMoments* as the first step instead of *vtkComputeMoments* in Figure 2.

5.1. Integration

The first option follows a straight forward integration. The communication is handled through MPI calls.

In order to provide the necessary information across nodes, the filter first computes the bounds of all blocks in the dataset and makes them known to all nodes using *MPI_AllGather()*. Then, each process performs four steps.

1. It computes the (potentially partial) moments for all points on this grid; this works analogously to the serial case. If part of the integration area lies outside the process' data, it will just add up the integral of the data it possesses.
2. It uses the bounds of the other processes to look where points close to its own boundary fall in the bounds of other processes and sends the locations over through *MPI_Send* and *MPI_Receive*. To avoid deadlocks between two rank, the one with the lower ID sends first, while the one with the higher ID receives first. The locations are encoded as the indices of the points in *vtkImageData* for each dimension. The ranges of possible indices in *vtkImageData* are stored in the variable "Extension," which is an integer array of length 6. Our encoding can produce indices outside the natural range of the receiving process because the location is outside its bounds. Still, the location can easily be reconstructed from *vtkImageData*'s parameters. Please note that it is not necessary to exchange the data across the nodes.
3. It computes the parts of the moments centered in other processes but intersecting its own domain just like the serial algorithm does and sends the results back to the owner as before through *MPI_Send* and *MPI_Receive*.
4. It adds up the native and incoming moment parts.

Given that the numerical integration is just a sum and therefore commutative, we get the true values after all processes have finished the computations on their areas.

5.2. FFT

For our parallel C/C++ FFT implementation, we use the *DFFTLIB* package, first developed and used in [GMB*14]. This library supports fast d-dimensional distributed FFT over MPI using a local FFT library. The local FFT functions have been abstracted in a way that allows adding new backends easily [Gla]. One such backend could be the *FFTW* package, a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data [FJ05]. *FFTW* is one of the most frequently used C/C++ FFT libraries, but for licensing issues while developing our algorithm for *VTK*, we opted to use the *KISSFFT* for serial and *DFFTLIB* for distributed parallel in our FFT implementation.

The parallel FFT algorithm carries out the padding similar to the serial FFT algorithm, but now it also needs to make sure that only the global boundaries, not local boundaries, are padded and that the padded input dataset is a power of 2. After the padding, the parallel algorithm has an additional step before FFT library function calls can be invoked and that is to redistribute the padded input dataset so that each rank has a same-sized block. Later, when the algorithm has truncated the output back to its original size, another redistribution is needed to get back to the original layout of the input data.

6. Experiments

In this section, we describe the datasets, the experimental setups, and their results. Because the computation of the moments are the bottleneck of the algorithm w.r.t. runtime and also the part that is not trivial to parallelize, we only analyze this step of the algorithm. We validated that the results of the two parallel approaches are identical to each other and their two serial counter parts up to numerical errors in the range of the accuracy of floating point precision. The following step that does the actual pattern detection would be identical for both approaches and the serial case, and is therefore not of interest for this study. We want to compare the two approaches w.r.t. runtime, scalability, and applicability.

6.1. Datasets

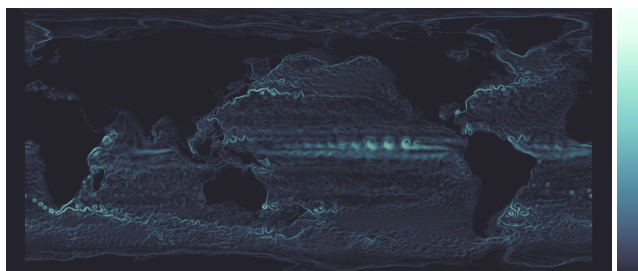


Figure 4: The geographic projection of MPAS-Ocean dataset. It is designed for the simulation of the ocean system from time scales of months to millenia and spatial scales from sub 1 km to global circulations [RPH*13]. The magnitude of the velocity is color coded.

In order to explore issues of scalability and choice of parameters on a range of scientific data, we chose three representative datasets

for our experiments. The first dataset is a single time step from an MPAS-Ocean simulation [RPH*13], as shown in Figure 4. The selected subset of MPAS-Ocean dataset that was presented as the example in Figure 2 is part of this dataset. For the experiments, we used the full MPAS-Ocean dataset rather than the subset in Figure 4. MPAS-Ocean is a simulation of the ocean system from time scales of months to millenia and spatial scales from sub 1 km to global circulations. The simulation is intended to facilitate the ability to reproduce mesoscale ocean activities so that scientists can study the multiscale anthropogenic climate change [RPH*13]. The MPAS-Ocean dataset that we used is a two dimensional dataset; we focused on the vector field named *velocity* in our experiments. For the parameter study, we used a dataset that has a resolution of 256^2 . For the scaling study, we used datasets that have resolutions ranging from 256^2 to 8192^2 . For reference, an MPAS-Ocean dataset with a resolution of 1024^2 is roughly 35 MB with 1,048,576 points.

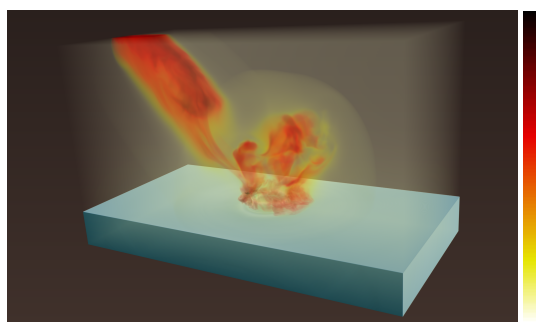


Figure 5: The volume rendering of the yA31 asteroid dataset. The asteroid simulation focuses on the wave formation of the ocean surface after asteroid impact [PST*16]. The value of the temperature variable is color coded and the solid blue is the ocean.

The second dataset is a single time step taken from the yA31 asteroid simulation [PST*16], as shown in Figure 5. This time step is a dataset produced by the simulation of an asteroid impact on deep ocean water using xRage [GWC*08]. This simulation is intended to imitate the deep water reactions to asteroid impacts so that scientists can study the likeliest danger level of an asteroid impact given different asteroid compositions, size, impact height, and entry angle. The simulation ran for 476 time steps in total, and the single time step that we used is one of the time steps after the asteroid impact [PST*16]. The asteroid dataset that we used is a three-dimensional dataset; we focused on the scalar field named *tev* in our experiments. For the parameter study, we used a dataset that has a resolution of $64 \times 64 \times 32$. For the scaling study, we used datasets that have resolutions ranging from $64 \times 64 \times 32$ to 512^3 . As a reference, a yA31 asteroid dataset that has a resolution of 128^3 is roughly 135 MB with 2,097,152 points.

The third dataset is a single time step taken from the NGC [DH16] Nyx cosmology simulation [ABL*13], as shown in Figure 6. This time step is a dataset produced by the simulation of the evolution of a system of discrete dark matter particles gravitationally coupled to an inviscid ideal fluid in an expanding universe. The simulation is intended to replicate large-scale dark matter activities so that scientists can study the cosmological behaviors [ABL*13]. The Nyx cosmology dataset that we used is a three-dimensional dataset;

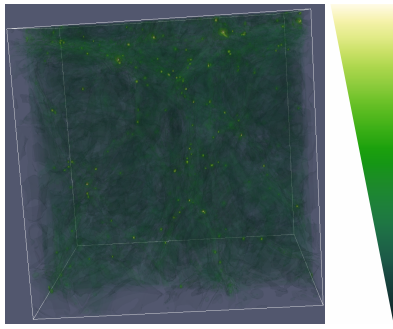


Figure 6: The volume rendering of the Nyx cosmology dataset. The cosmology simulation follows the temporal evolution of a system of discrete dark matter particles gravitationally coupled to an inviscid ideal fluid in an expanding universe [ABL*13]. The value of the density variable is color coded.

we focused on the scalar field named *dark matter density* in our experiments. For the parameter study, we used a dataset that has a resolution of $64 \times 64 \times 32$. For the scaling study, we used datasets that have resolutions ranging from $64 \times 64 \times 32$ to 512^3 . As a reference, a Nyx cosmology dataset that has a resolution of 128^3 is roughly 182 MB and 2,097,152 points.

6.2. Experimental Setup

All the computations are performed on NERSC’s Cori supercomputer (<https://docs.nersc.gov/systems/cori/>) using the Intel Xeon “Haswell” processors. Each node has two sockets, each socket is populated with a 16-core Intel Xeon Processor E5-2698 v3 (“Haswell”) at 2.3 GHz. Each node has 128 GB DDR4 2133 MHz memory (four 16-GB DIMMs per socket). For the experiments, each physical core is tasked with at most one MPI process, so if we want to run 1024 MPI tasks, we would use 32 nodes as each node has 32 physical cores. For better accuracy, the timing measurements are computed as an average of at least 10 runs with the same parameters on the same dataset, except for the large convolution runs. Please note all presented timings were acquired post hoc to avoid tainting the results with runtimes of the simulations. Proof that our algorithms are in situ ready will be presented in Section 7.

We first present a study of the impact of the most important algorithm parameters on the run time. We use the three datasets as described in Section 6.1, with five different resolutions, orders of basis functions ranging from 0 to 5, and 5 different radii. The results are found in Figures 7, 8, and 9. We have presented only the serial results because the parallel results exhibit the same behavior for all the parameter studies.

Then, we present the scaling study for the three datasets from Section 6.1. For the scaling study, we use representative values for the order and integration radii parameters that are kept constant across runs. We vary the number of MPI ranks up to 1024. The different run times for weak and strong scaling are shown in Figures 10 and 11.

6.3. Parameter Study

The major inputs to the pattern detection algorithm for both approaches are the dataset and the pattern that is to be searched within the dataset. An example flowchart is shown in Figure 2. In addition to these major inputs, there are three parameters that affect the computation and performance of the algorithm significantly. The results for these parameters—the resolution, the order of the basis function, and the radius—are shown in Figures 7, 8, and 9, respectively.

Consider first the resolution on which the moments are computed. In a data-distributed environment, the storage capacity is limited. Further, just like a Taylor decomposition, the moments contain information about the underlying function in the region of their center. Thus, we often do not need to compute the moments at every point of the original dataset, but only on a subset. Note that the full resolution is still used for the computation of the moments at each of the chosen points. A coarser resolution will allow the integration approach to save computation time as fewer operations are needed. However, the FFT approach will not benefit from a coarser resolution as we need to transform the whole dataset no matter how much of the available information will finally be stored.

The second parameter is the order up to which the basis functions are used for building the kernel. Theoretically, there are infinitely many moments and the function can be restored exactly, if we could consider all the information. Practically, we will cut off the computation at a given order. The remaining information is an approximation to the original function analogous to a Taylor series.

The final parameter is the radius used to build the stencil for the pattern matching. The radius parameter is the product of the spacing given in the dataset and the number of steps in each dimension. This radius corresponds to the feature size for which we are searching. In real applications, that radius will usually be a range of different sizes corresponding to features (patterns) with physical meaning.

We designed the experiments to cover a relevant span of the parameter space. The resolution parameter describes the total number of points on which the moments will be computed. The resolution experiments range from 100% resolution (the total number of points that the original dataset possesses), down to 6.25% resolution, where we reduce the resolution by half each time. For the order parameter, we ran experiments with basis functions going up to order 5. For the radius parameter, we examined a maximum radius of one half of the minimum bound in all dimensions of the dataset. The radius parameter is reduced by one half each time for the next radius, and this reduction is done four times.

As we can observe from the parameter study graphs, when the order of basis functions is increased (Figure 8), the run time of both approaches grows. Note, however, that the difference between the two approaches increases as the order increases. Given any higher order of basis functions or larger datasets, the FFT approach will outperform the integration approach. In Figure 9, we can see that the performance of the FFT approach is not affected by the size of the radius for the stencil, whereas the performance time of the integration approach does increase noticeably. On the other hand, Figure 7 shows that, as expected, the integration approach benefits from computing the moments on only a subset of the original data whereas the FFT approach remains unaffected by this radius size.

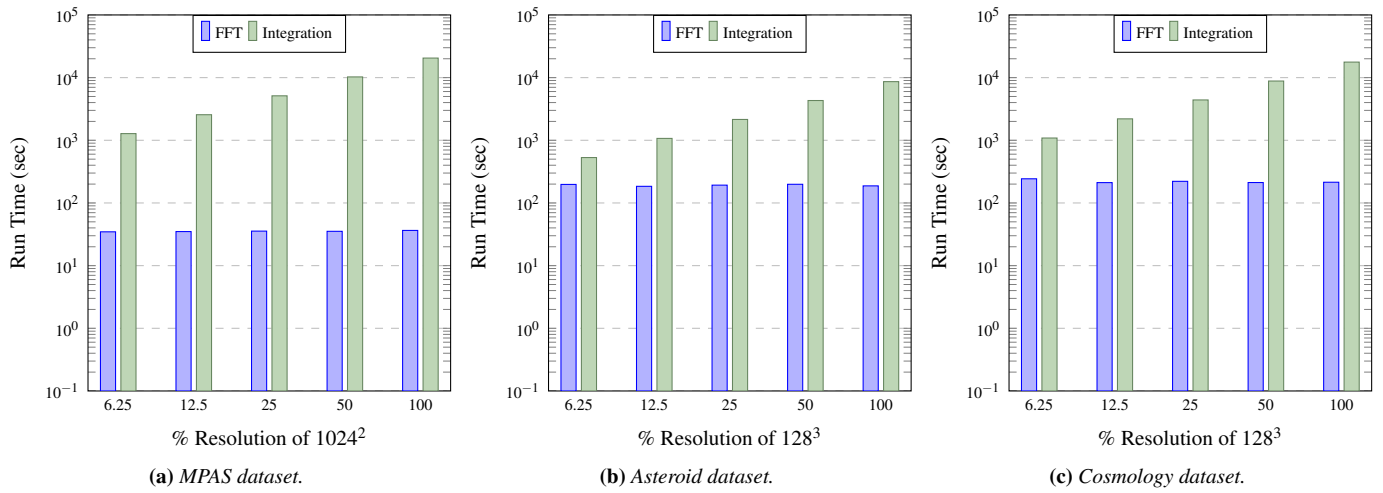


Figure 7: Performance in seconds for varying the grid resolution using the three datasets from Section 6.1 with order 2 and radius ratio 1/16.

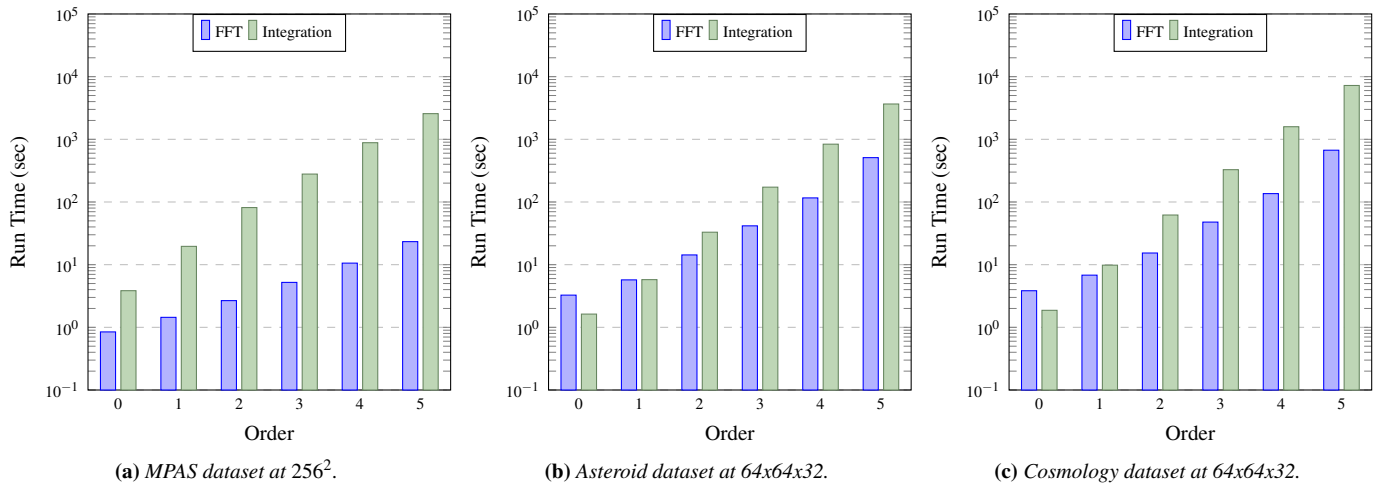


Figure 8: Performance in seconds for varying the order parameter for the three datasets from Section 6.1 with radius ratio 1/16.

Notice that in the performance graphs for all the datasets, the FFT approach demonstrates better performance than the integration approach in the majority of the experiments. It is only when the resolution is extremely low, or reasonably small and combined with low order or radius, that the integration approach outperforms the FFT approach. However, scientists are not likely to choose an approach that limits their output data to such a low resolution, order, or radius, so the FFT approach will generally be the preferred option.

6.4. Scaling Study

Scalability is the capability of the algorithm to handle an increasing amount of work. An algorithm is considered to be scalable if it is suitably efficient and practical when it is applied to large data computations. The amount and the size of output data from simulations are growing along with the growth of supercomputers' capabilities

and capacities. Therefore, scalable algorithms gain importance as more and more large datasets are being produced. Consequently, scalability is the focus of this next study. The different run times for weak and strong scaling of our algorithm with the MPAS-Ocean, asteroid, and cosmology datasets are shown in Figures 10 and 11.

For the weak scaling study, the size of data given to each processor is constant and additional processors are added accordingly as the data size increases. As a result, this type of measurement is used as justification for algorithms that take a lot of memory and system resources, or are memory bound. In the case of weak scaling, linear scaling is achieved if the run time stays constant while the data size is increased in direct proportion to the number of processors.

For the strong scaling study, the size of the data is fixed but the number of processors is increased. As a result, this type of measurement is used as justification for algorithms that run for a

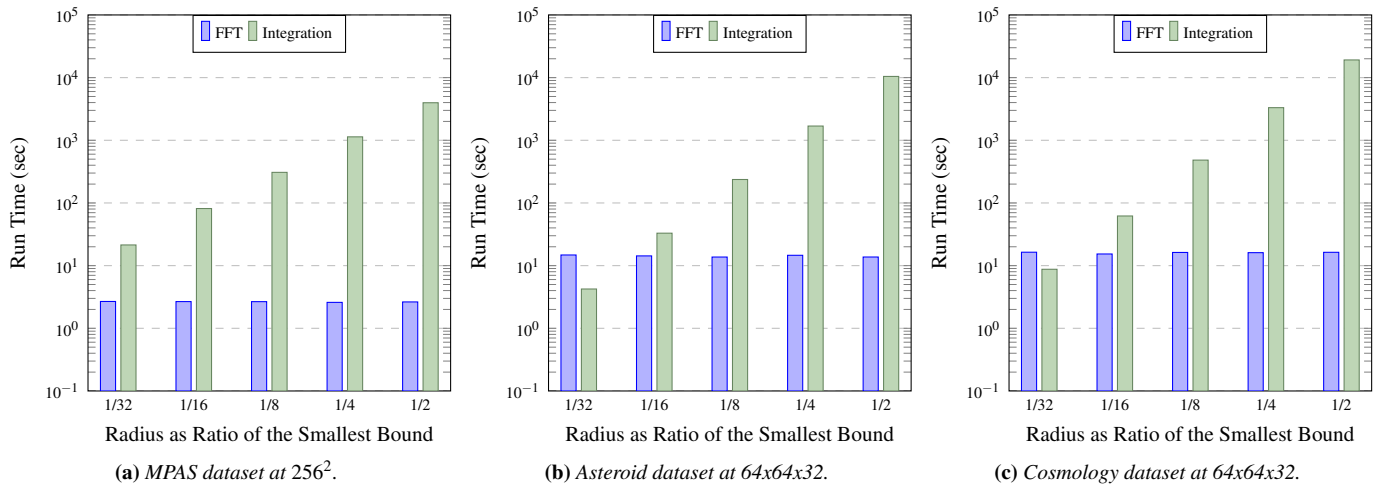


Figure 9: Performance in seconds for varying the radius parameter using the three datasets from Section 6.1 with order 2.

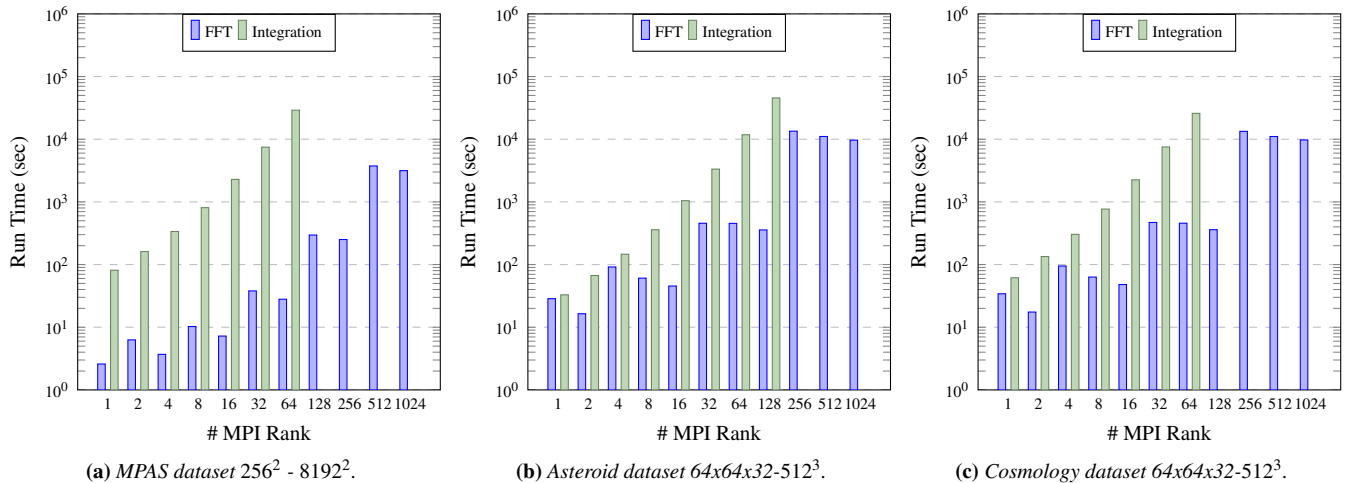


Figure 10: Performance in seconds for weak scaling using the three datasets from Section 6.1 with order 2 and constant radius.

long time or are CPU bound. In the case of strong scaling, linear scaling is achieved if the speedup factor is equal to the number of processors used.

For both scaling studies, the experiments are run on numbers of MPI ranks from 1 to 1024. All three datasets, MPAS-Ocean, asteroid, and cosmology, are run with the order parameter set to 2 and the radius parameter set to a representative value. The spacing is also kept constant in order to have a constant kernel size. The weak scaling for cosmology and asteroid uses the dataset that is a resampled resolution of $64 \times 64 \times 32$ all the way up to a resampled resolution of 512^3 . The weak scaling for the MPAS-Ocean uses the dataset that is a resampled resolution of 256^2 up to 8192^2 . The strong scaling for cosmology and asteroid uses the dataset that is a resampled resolution of 128^3 . The strong scaling for MPAS-Ocean uses the dataset that is a resampled resolution of 1024^2 .

Looking at the weak scaling experiment in Figure 10, we see that the run time of the integration approach increases exponentially whereas the run time of the FFT approach demonstrates better scalability with overall lower values and a moderately slower rate of increase. On top of that, the larger configuration runs for the integration approach of moments computation did not finish running even given a job wall time of 24 hours with all the datasets. Therefore, the FFT approach, choosing realistic parameter combinations, offers the ability to run in a more reasonable time given the necessary number of ranks than the integration approach. Notice that the run time for the FFT approach increases over time, but with a certain wave-like behavior. This behavior is because the algorithm of the FFT approach is very dependent on the geometry of the input dataset and the processors' grid. In particular, the FFT approach prefers square datasets in two dimensions or cube datasets in three dimensions.

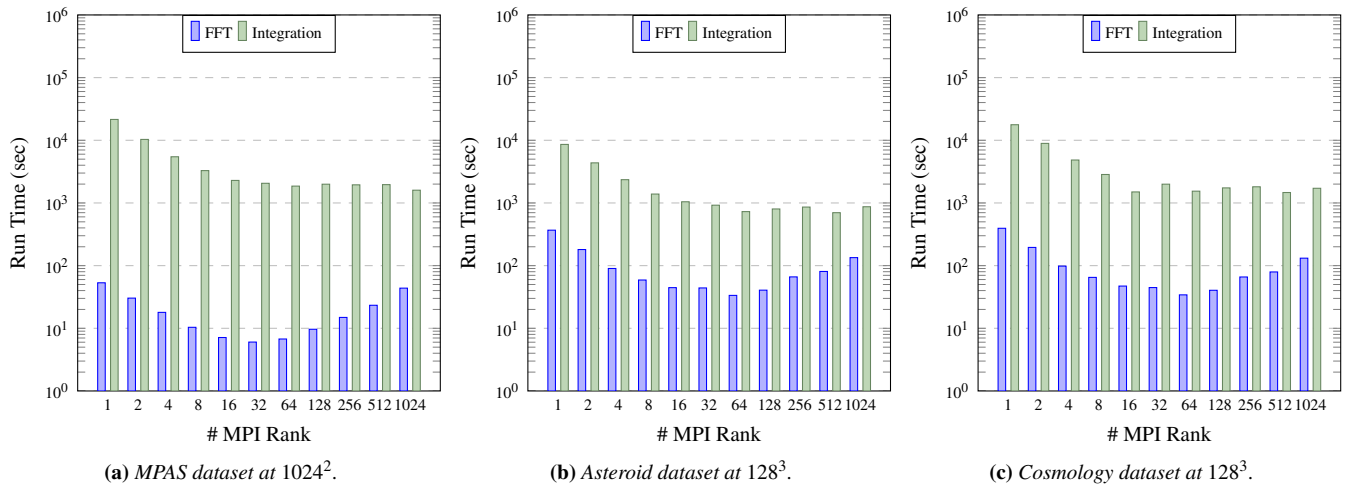


Figure 11: Performance in seconds for strong scaling using the three datasets from Section 6.1 with order 2 and constant radius.

The FFT approach demonstrates relatively better weak scalability when compared to the integration approach, but the FFT approach will not perform as well when compared to other algorithms that employ a nearest-neighbor communication pattern. Because the FFT approach employs heavy global communications for its computations such as transposes, the communication overhead increases in proportion to the number of processors, whereas the algorithms that employ nearest-neighbor communication patterns have relatively constant communication overhead.

Looking at the strong scaling experiments in Figure 11, we see that initially the overall rate at which the run time of the FFT approach speeds up is slightly better than, though very similar to, the rate at which the integration approach speeds up as the number of processors increases. While the rate at which the integration approach flattens out as its communications overhead start to offset the benefit of having more processors share the work, the communication overhead actually starts outweighing the benefit of having more processors share the work in the case for the FFT approach. Although the strong scalability of the integration approach is better later on, the FFT approach still offers better actual performance.

As discussed above, the communication overhead for the FFT approach increases relative to the number of processors, and this increase is more noticeable for the FFT approach than for the integration approach, so it is harder for the FFT approach than for the integration approach to achieve strong scaling. The goal in the strong scaling study is to find the best configuration, 64 ranks in this particular case, that allows the moment computations to complete in a reasonable amount of time, yet does not waste too many cycles caused by parallel overhead, such as global communications. As a result, even though the FFT approach does not offer as effective of a strong scaling as the integration approach, the FFT approach will still be a more realistic option given its faster physical run time.

7. In Situ Integration

As example of the in situ application of our algorithm, we demonstrate an end-to-end integration with two real world ECP applications MFIX-Exa [MEb, MEa] and WarpX [VAB*18].

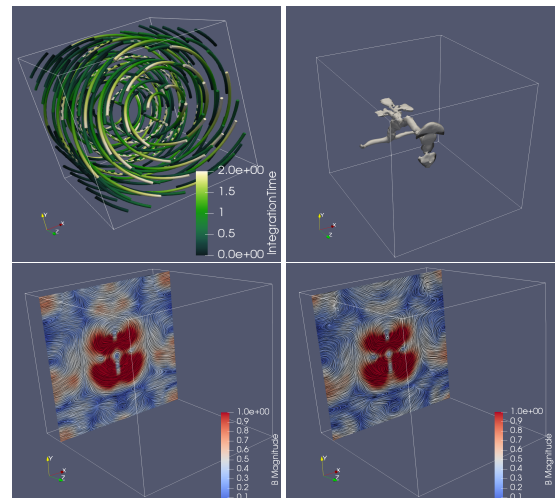


Figure 12: (Left) The pattern is a vortex. (Right) in situ visualization with reduced occlusion through in situ rotation-invariant pattern detection of the magnetic field of the WarpX simulation.

Our first example shows the application of our algorithm for an in situ visualization task with WarpX [VAB*18], which is a simulation modeling particle accelerators. We use our algorithm after integrating the in situ infrastructure with WarpX (<https://warpx.readthedocs.io/en/latest/visualization/ascent.html>) on the OLCF machine Summit (<https://www.olcf.ornl.gov/summit/>) through Ascent [Asc], which provides a general integration with ParaView [AGL05]. Once, the connection to Ascent is established on the simulation

side, it allows importing the data in a VTK format and calling VTK filters as modules integrated into ParaView through a ParaView trace in python (https://www.paraview.org/Wiki/ParaView_and_Python).

We use the pattern detection to visualize vortical behavior in the magnetic field using the vortex pattern in Figure 12 left. The right image shows the isosurface of the largest connected component detected by our algorithm. This isosurface shows a vortex core that branches off several times. As reference, we show the magnetic field using line integral convolution (LIC) right before (third from left) and right after (right) the biggest branching occurs. It can be verified that the central vortex indeed splits into two cores.

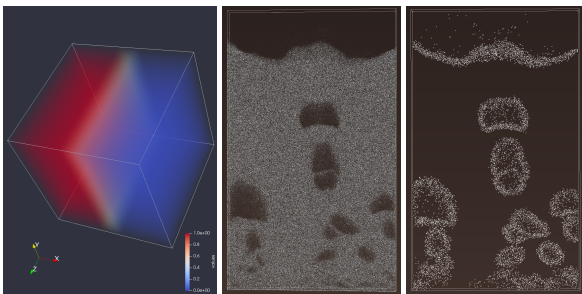


Figure 13: (Left) The pattern is a step function. (Middle) The original dataset of MFIX-Exa bubble bed simulation with particles rendered as spheres. (Right) The dataset reduced to 0.5% of its original size to only the bubble boundaries through in situ rotation-invariant pattern detection.

Our second example showcases the application of our algorithm for data reduction to a real world bubble bed simulation running with MFIX-Exa, which is a computational fluid dynamics—discrete element model (CFD-DEM) code developed for efficient runs on current and next-generation massively parallel supercomputing architectures [MEb, MEa]. Our code was successfully integrated with the simulation and ran in situ on NERSC’s Cori (<https://docs.nersc.gov/systems/cori/>). MFIX-Exa does not have an Ascent or Catalyst [ABG*15] integration yet. We therefore built MFIX-Exa with the VTK libraries linked in. The conversion of the raw particle data into the VTK data that contains the particle density field and the setup of the VTK pipeline of our workflow were done explicitly in the MFIX-Exa classes and AMReX_Particles.H and AMReX_ParticleContainerI.H.

We identify regions of interest using the moment invariants algorithm in situ and then subsample the original particle dataset at a coarser resolution for regions of less interest before saving any data to disk. In this example, we have used a step function as a pattern on the particle density field because we know from the application scientists that they are particularly interested in the boundaries of the bubbles. As the discarded regions had a very homogeneous behavior, the loss of information in the final output is small even though the reduction is tremendous, Figure 13.

8. Discussion and Conclusion

We have successfully presented a moment invariants algorithm available through VTK in the distributed data environment and demon-

strated the acceleration of the algorithm through the FFT and the cross-correlation theorem. In addition, we conducted and discussed experimental results from a relevant parameter study and a scaling study of both, the integration approach and the FFT approach. Furthermore, we have showcased the benefit in data reduction with the bubble bed simulation MFIX-Exa by subsampling according to the result we get from the moment invariants output and its in situ readiness with both simulations by integrating the moment invariants analysis into the simulation codes and running them in situ.

Both approaches successfully generalized the original moments computation algorithm to run in a distributed setting. The results are identical to the serial run up to numerical errors.

Overall, the FFT approach to moment computation has been shown as the preferred option because of the great run time performance with different input parameter combinations. That approach shows that the favorable local communication pattern of the integration approach does not compensate its computational overhead. However, the FFT approach requires structured datasets and has certain assumptions, or limitations, in the area of expected data distribution. In particular, the *DFTLIB* expects the data to be distributed regularly across all the ranks. As in situ algorithms could have difficulties influencing the structure or the distribution of the dataset, this colorredrequirement makes the integration approach preferred when non-structured datasets are used or irregular distributions arise. Further, the FFT approach needs more storage to pad the data, which may cause issues in an in situ setting where resources are scarce. Therefore, we conclude that the best option is a combination of using the FFT approach whenever applicable and providing the option of the integration approach in the other cases.

As future work, we plan on optimizing and integrating the moments filter into *VTK-m* where they will become GPU-enabled algorithms. We are also planning on implementing the algorithms that we have developed in *VTK* directly into ParaView to make them more readily available and allow easier access to users.

9. Acknowledgements

We would like to thank Li-Ta Lo, Boonthanome Nouanesengsy, and David Daniel for their integral support and advice with regard to code development in a distributed environment, and Mark Borgering and Jens Glaser for the support on their FFT libraries. We would also like to thank Jan Flusser, Francesca Samsel, Jonathan Woodring, Wendy Caldwell, and Terece Turton for their support on this project. This work is supported by the Exascale Computing Project through ALPINE. and released under LA-UR-17-27865.

References

- [ABG*15] AYACHIT U., BAUER A., GEVECI B., O’LEARY P., MORELAND K., FABIAN N., MAULDIN J.: Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (2015), pp. 25–29. 2, 10
- [ABL*13] ALMGREN A. S., BELL J. B., LIJEWSKI M. J., LUKIÄĞ Z., ANDEL E. V.: Nyx: A massively parallel amr code for computational cosmology. *The Astrophysical Journal* 765, 1 (2013), 39. 5, 6
- [AGL05] AHRENS J., GEVECI B., LAW C.: Paraview: An end-user tool for large data visualization. *The visualization handbook* 717 (2005). 9

- [Ase] ASCENT: A flyweight in situ visualization and analysis runtime for multi-physics hpc simulations - alpine-dav/ascent. <https://github.com/alpine-dav/ascent>. 2, 9
- [AWW*16] AYACHIT U., WHITLOCK B., WOLF M., LORING B., GEVECI B., LONIE D., BETHEL E. W.: The sensei generic in situ interface. In *2016 Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)* (2016), IEEE, pp. 40–44. 2
- [BAA*16] BAUER A. C., ABBASI H., AHRENS J., CHILDS H., GEVECI B., KLASKY S., MORELAND K., O'LEARY P., VISHWANATH V., WHITLOCK B., ET AL.: In situ methods, infrastructures, and applications on high performance computing platforms. In *Computer Graphics Forum* (2016), vol. 35, Wiley Online Library, pp. 577–597. 2
- [BH17] BUJACK R., HAGEN H.: Moment Invariants for Multi-Dimensional Data. In *Modelling, Analysis, and Visualization of Anisotropy* (2017), Ozerlan E., Schultz T., Hotz I., (Eds.), Mathematica and Visualization, Springer Basel AG. 2, 3, 4
- [BHS15] BUJACK R., HOTZ I., SCHEUERMANN G., HITZER E.: Moment Invariants for 2D Flow Fields via Normalization in Detail. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 21, 8 (Aug 2015), 916–929. 2
- [BKH*15] BUJACK R., KASTEN J., HOTZ I., SCHEUERMANN G., HITZER E.: Moment invariants for 3d flow fields via normalization. In *Visualization Symposium (PacificVis), 2015 IEEE Pacific* (2015), IEEE, pp. 9–16. 2
- [Bor] BORGERDING M.: Kissfft library. <https://github.com/mborgerding/kissfft>. 3
- [BRA18] BUJACK R., ROGERS D., AHRENS J.: Reducing Occlusion in Cinema Databases through Feature-Centric Visualizations. In *Leipzig Symposium on Visualization In Applications (LEVIA)* (2018). 2
- [Che90] CHEN K.: Efficient parallel algorithms for the computation of two-dimensional image moments. *Pattern Recognition* 23, 1-2 (1990), 109–119. 2
- [CPA*10] CHILDS H., PUGMIRE D., AHERN S., WHITLOCK B., HOWISON M., PRABHAT, WEBER G., BETHEL E. W.: Extreme Scaling of Production Visualization Software on Diverse Architectures. *IEEE Computer Graphics and Applications (CG&A)* 30, 3 (May/June 2010), 22–31. 1
- [DCH*16] DUTTA S., CHEN C.-M., HEINLEIN G., SHEN H.-W., CHEN J.-P.: In situ distribution guided analysis and visualization of transonic jet engine simulations. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 811–820. 2
- [DH16] DANIEL D. J., HUNGERFORD A. L.: *LANL ASC Advanced Technology Development and Mitigation: Next-Generation Code project (NGC)*. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2016. 5
- [DN77] DIRILTEN H., NEWMAN T. G.: Pattern matching under affine transformations. *Computers, IEEE Transactions on* 100, 3 (1977), 314–317. 2, 3
- [FJ05] FRIGO M., JOHNSON S. G.: The design and implementation of fftw3. In *PROCEEDINGS OF THE IEEE* (2005), pp. 216–231. 5
- [Flu00] FLUSSER J.: On the independence of rotation moment invariants. *Pattern Recognition* 33, 9 (2000), 1405–1410. 2
- [FSZ16] FLUSSER J., SUK T., ZITOVÁ B.: *2D and 3D Image Analysis by Moments*. John Wiley & Sons, 2016. 2
- [Gla] GLASER J.: Dfftilib library. <https://github.com/jglaser/dfftilib>. 5
- [GMB*14] GLASER J., MEDAPURAM P., BEARDSLEY T. M., MATSEN M. W., MORSE D. C.: Universality of block copolymer melts. *Phys. Rev. Lett.* 113 (Aug 2014), 068302. 5
- [GWC*08] GITTINGS M., WEAVER R., CLOVER M., BETLACH T., BYRNE N., COKER R., DENDY E., HUECKSTAEDT R., NEW K., OAKES W. R., ET AL.: The rage radiation-hydrodynamic code. *Computational Science & Discovery* 1, 1 (2008), 015005. 5
- [HSS11] HOSNY K. M., SHOUMAN M. A., SALAM H. M. A.: Fast computation of orthogonal fourier–mellin moments in polar coordinates. *Journal of Real-Time Image Processing* 6, 2 (2011), 73–80. 2
- [Hu62] HU M.-K.: Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory* 8, 2 (1962), 179–187. 2
- [Kin76] KINTNER E. C.: On the mathematical properties of the zernike polynomials. *Journal of Modern Optics* 23, 8 (1976), 679–680. 2
- [Kit] KITWARE: vtk module momentinvariants. <https://gitlab.kitware.com/vtk/momentinvariants>. 2
- [KPZ11] KUHLEN T., PAJAROLA R., ZHOU K.: Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization (EGPGV)* (2011), vol. 10, Eurographics Association Aire-la-Ville, Switzerland, pp. 101–109. 2
- [Lan14] LANGBEIN M.: *Higher Order Moment Invariants and their Applications*. PhD Dissertation, Technical University Kaiserslautern, 2014. 2
- [LH09] LANGBEIN M., HAGEN H.: A Generalization of Moment Invariants on 2D Vector Fields to Tensor Fields of Arbitrary Order and Dimension. In *Advances in Visual Computing*, Bebis G., Boyle R., Parvin B., Koracin D., Kuno Y., Wang J., Pajarola R., Lindstrom P., Hinkenjann A., Encarnaçãõ M., Silva C., Coming D., (Eds.), vol. 5876 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 1151–1160. 2, 3
- [lib] LIBSIM: Visit tutorial in situ. <http://visitusers.org/index.php?title=visit-tutorial-in-situ>. 2
- [LZKS09] LOFSTEAD J., ZHENG F., KLASKY S., SCHWAN K.: Adaptable, metadata rich io methods for portable high performance io. In *2009 IEEE International Symposium on Parallel & Distributed Processing* (2009), IEEE, pp. 1–10. 2
- [MEa] MFiX-EXA: Mfix-exa: Optimizing a new technology to reduce power plant carbon dioxide emissions. <https://mfix.netl.doe.gov/mfix-exa-optimizing-a-new-technology-to-reduce-power-plant-carbon-dioxide-emissions/>. 2, 9, 10
- [MEb] MFiX-EXA: Optimizing a new technology to reduce power plant carbon dioxide emissions. <https://www.exascaleproject.org/optimizing-a-new-technology-to-reduce-power-plant-carbon-dioxide-emissions/>. 2, 9, 10
- [MR95] MUKUNDAN R., RAMAKRISHNAN K.: Fast computation of legendre and zernike moments. *Pattern recognition* 28, 9 (1995), 1433–1442. 2
- [PCO85] PINJO Z., CYGANSKI D., ORR J. A.: Determination of 3-D object orientation from projections. *Pattern Recognition Letters* 3, 5 (1985), 351–356. 2, 3
- [PKK08] PAPA KOSTAS G. A., KARAKASIS E. G., KOULOURIOTIS D. E.: Efficient and accurate computation of geometric moments on gray-scale images. *Pattern Recognition* 41, 6 (2008), 1895–1904. 2
- [PST*16] PATCHETT J. M., SAMSEL F., TSAI K. C., GISLER G. R., ROGERS D. H., ABRAM G., TURTON T. L.: Visualization and analysis of threats from asteroid ocean impacts. 5
- [RCMS12] RIVI M., CALORI L., MUSCIANISI G., SLAVNIC V.: In-situ visualization: State-of-the-art and some use cases. *PRACE White Paper* (2012), 1–18. 2
- [RPH*13] RINGLER T., PETERSEN M., HIGDON R. L., JACOBSEN D., JONES P. W., MALTRUD M.: A multi-resolution approach to global ocean modeling. *Ocean Modelling* 69 (2013), 211–232. 5
- [SB11] SPILIOIOTIS I. M., BOUTALIS Y. S.: Parameterized real-time moment computation on gray images using block techniques. *Journal of Real-Time Image Processing* 6, 2 (2011), 81–91. 2
- [SF11] SUK T., FLUSSER J.: Tensor method for constructing 3d moment invariants. In *Computer Analysis of Images and Patterns* (2011), Springer, pp. 212–219. 2

- [SH80] SADJADI F. A., HALL E. L.: Three-Dimensional Moment Invariants. *IEEE Trans. Pattern Anal. Mach. Intell. PAMI-2*, 2 (1980), 127–136. [2](#)
- [SHF12] SUK T., HÖSCHL C., FLUSSER J.: Decomposition of binary images—a survey and comparison. *Pattern Recognition* 45, 12 (2012), 4279–4291. [2](#)
- [SHM*07] SCHLEMMER M., HERINGER M., MORR F., HOTZ I., HERING-BERTRAM M., GARTH C., KOLLMANN W., HAMANN B., HAGEN H.: Moment Invariants for the Analysis of 2D Flow Fields. *IEEE Transactions on Visualization and Computer Graphics* 13, 6 (2007), 1743–1750. [2](#)
- [SLM04] SCHROEDER W. J., LORENSEN B., MARTIN K.: *The visualization toolkit: an object-oriented approach to 3D graphics*. Kitware, 2004. [2](#)
- [VAB*18] VAY J.-L., ALMGREN A., BELL J., GE L., GROTE D., HOGAN M., KONONENKO O., LEHE R., MYERS A., NG C., ET AL.: Warp-x: A new exascale computing platform for beam–plasma simulations. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 909 (2018), 476–479. [2](#), [9](#)
- [WPS*15] WOODRING J., PETERSEN M., SCHMEIßER A., PATCHETT J., AHRENS J., HAGEN H.: In situ eddy analysis in a high-resolution ocean climate model. *IEEE transactions on visualization and computer graphics* 22, 1 (2015), 857–866. [2](#)
- [YA96] YANG L., ALBREGTSEN F.: Fast and exact computation of cartesian geometric moments using discrete green’s theorem. *Pattern Recognition* 29, 7 (1996), 1061–1073. [2](#)
- [YWG*10] YU H., WANG C., GROUT R. W., CHEN J. H., MA K.-L.: In situ visualization for large-scale combustion simulations. *IEEE computer graphics and applications* 30, 3 (2010), 45–57. [2](#)