A Model for Optimizing File Access Patterns using Spatio-Temporal Parallelism

Boonthanome Nouanesengsy
Los Alamos National Laboratory
boonth@lanl.gov

John Patchett
Los Alamos National Laboratory
patchett@lanl.gov

James Ahrens
Los Alamos National Laboratory
ahrens@lanl.gov

Andrew Bauer
Kitware Inc.
andy.bauer@kitware.com

Aashish Chaudhary
Kitware Inc.
aashish.chaudhary@kitware.com

Ross Miller
Oak Ridge National Laboratory
rgmiller@ornl.gov

Berk Geveci
Kitware Inc.
berk.geveci@kitware.com

Galen M. Shipman
Oak Ridge National Laboratory
gshipman@ornl.gov

Dean N. Williams
Lawrence Livermore National Laboratory
williams13@llnl.gov

ABSTRACT

For many years now, I/O read time has been recognized as the primary bottleneck for parallel visualization and analysis of large-scale data. In this paper, we introduce a model that can estimate the read time for a file stored in a parallel filesystem when given the file access pattern. Read times ultimately depend on how the file is stored and the access pattern used to read the file. The file access pattern will be dictated by the type of parallel decomposition used. We employ spatio-temporal parallelism, which combines both spatial and temporal parallelism, to provide greater flexibility to possible file access patterns. Using our model, we were able to configure the spatio-temporal parallelism to design optimized read access patterns that resulted in a speedup factor of approximately 400 over traditional file access patterns.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; D.1.3 [**Concurrent Programming**]: Parallel programming

Keywords

Visualization, Data Analysis, I/O, Modeling, Parallel Techniques

1. INTRODUCTION

The visualization and analysis of large-scale data in a timely manner has been a recognized problem for many years now. With computational power increasing and the introduction of more sensitive instrumentation, data from both simulations and experiments are expected to continue to grow. The result of these trends are ever larger datasets containing higher spatial and temporal resolutions. The standard method to tackle the large-data problem is to use parallel processing. In practice, the main bottleneck in large-scale parallel analysis is the I/O read step. One of the main factors in I/O performance is how the file is accessed, and what the read pattern is. The chosen parallel decomposition strategy determines the read pattern.

For many of the common visualization tools used by the community, parallel processing implies using a data-parallel approach employing only spatial parallelism. In this approach, the data is partitioned spatially and spread out across several processes. Each process then applies the same computation on their piece of data. Another option for data decomposition, temporal parallelism, involves processing data from different timesteps simultaneously. The same computations are applied to each timestep. Often overlooked is how different decomposition approaches affect the read pattern in files, despite the fact that I/O performance is usually the performance bottleneck.

In this paper, we contribute a model which estimates the total running time of a parallel visualization and analysis pipeline. We also contribute the implementation of a pipeline capable of spatio-temporal parallelism, called the spatio-temporal pipeline, in a general purpose visualization tool. Our model can estimate the time needed to load a file stored in a parallel filesystem when given the file access pattern. The goal of our model is to provide a useful way of comparing various workflow choices, rather than accurately predicting exact running times. Using this model, we can determine which parallel approaches are best suited for a given file format and pipeline. To offer more control over the file access pattern, we implement a method that employs both spatial and temporal parallelism, called the spatio-temporal pipeline. In the spatio-temporal pipeline,

UltraVis'13, November 17 - 21 2013, Denver, CO, USA

http://dx.doi.org/10.1145/2535571.2535593

processes are separated into groups called time compartments. Temporal parallelism is employed as time compartments independently process a timestep concurrently. Each timestep is partitioned spatially over all processes within the time compartment, thus the spatio-temporal pipeline uses both spatial and temporal parallelism. The spatio-temporal pipeline is a fully integrated feature in ParaView and UV-CDAT (Ultrascale Visualization - Climate Data Analysis Tools). ParaView is a general-purpose parallel visualization tool, and UV-CDAT [13] is a visualization and analysis tool specializing in large-scale climate-data analysis.

Time-varying data can be stored in a myriad of different formats. How the data is stored and what access patterns are used to read in the data play a critical role in I/O read performance. In this paper, we focus our discussion to a very common file format: the time-varying data is composed of files, and each file represents one scalar field at one timestep.

The performance differences between temporal, spatial, and spatio-temporal parallelism may not be readily apparent. Indeed, if all components of a parallel system were to scale perfectly, there should be *no difference in running time* between using spatial parallelism, temporal parallelism, and spatio-temporal parallelism, for equal amounts of parallelization. In practice, there is a difference between various parallelization schemes because each method creates a distinct read access pattern, affecting the I/O read time. Ultimately, using spatio-temporal parallelism allows for greater flexibility in selecting a file access pattern that will provide greater I/O performance. The model is used to determine which file access pattern maximizes performance.

2. RELATED WORK

Attempts have been made to address I/O performance for visualization of large datasets. Some have improved I/O performance for post processing by using systems connected to faster storage such as solid state drives (SSDs) [3][8]. Mitchell *et al* [7], using VisIO, realized performance gains over the traditional high performance parallel file system by extending the ParaView system to support the Hadoop file system. Preprocessing data before post-processing has been shown capable of lessening I/O bandwidth requirements. Woodring *et al* [15] encode raw data in a JPEG 2000 format to enable multi-resolution streaming over low bandwidth connections. In-situ, in-transit, and hybrid combinations of these two paradigms have been used to lessen the necessity of post-processing, mitigating associated I/O issues [5][11][12].

Parallel post-processing of climate data is of major concern. Woitaszek *et al* [14] gained performance by parallelizing a post-processing workflow for climate data using the Swift scripting language, with parallelism only scaling to 32 processes. On the other hand, our work is focused primarily on I/O performance scaling to thousands of processes.

Both spatial and temporal parallelism have been studied previously. Yu *et al* [16][17] use a spatio-temporal scheme to achieve high I/O performance, but does not model the read step. Childs *et al* [4] showed through a series of large parallel visualization experiments that pure parallelism analysis operations work at extreme scales, but I/O times became very large, suggesting that the I/O performance required further study. The TECA project [10] reports good temporal parallelism performance for climate data, but has little parallel I/O or native spatial parallelism support, which creates a problem for data with large spatial bounds. While spatio-temporal parallelism is not new, our work focuses on the connection between the resulting read access pattern and the I/O performance. Our model explains why end-to-end scaling of pure spatial or pure temporal pipelines are sub-optimal.

Work has been performed to enable developers and end users the ability to alter data decompositions and thus affect read times. Kendall *et al* [6], using their BIL (Block I/O Layer) software, show considerable performance gains by aggregating smaller requests into larger requests which cover more contiguous regions of the file on disk (two-phase collective I/O). Peterka *et al* [9] provide a generalized library for building visualization algorithms on top of configurable domain decompositions, impacting I/O access patterns which are executed using a variety of library access methods, including BIL. Our contribution of a model will help users of these tools and systems make more wise data decomposition decisions.

Biddiscombe *et al* [2] introduced the concept of time to the ParaView pipeline. While this work enabled the serial processing of spatially decomposed time steps, our work enables decomposition in both space and time, allowing for the simultaneous processing of multiple time steps.

3. THE SPATIO-TEMPORAL PIPELINE

Spatial parallelism is a decomposition in which data is spatially partitioned over all available processes. Each process then applies the same set of computations on its piece of data. When employing spatial parallelism, timesteps are processed in serial, i.e. timestep 0 is processed, then timestep 1 is processed, etc. One side effect of spatial parallelism is that increasing the number of processes results in each timestep being spatially partitioned into more pieces, and each piece becomes smaller. According to the read model outlined in Section 4.5, this behavior adversely affects the read pattern and impairs I/O performance.

Temporal parallelism is a method in which multiple timesteps are processed in parallel. This is a form of pipeline parallelism, in which multiple pipelines are instantiated in order to process multiple inputs at once. Temporal parallelism usually requires large amounts of memory, as each process will load an entire timestep.

The spatio-temporal pipeline was designed to utilize both spatial and temporal parallelism, which allows for more control of the access pattern used to read files. Spatio-temporal parallelism is accomplished by first partitioning all available processes into groups called time compartments. Each time compartment is responsible for processing timesteps, and performs computations independently of each other. Each timestep is spatially partitioned over all processes within the time compartment. If there are more timesteps than time compartments, then a time compartment will process multiple timesteps. For example, if there are two time compartments and six timesteps, then each time compartment will process three timesteps. Each time compartment loads one timestep at a time, and when a timestep is finished the next available timestep is then loaded. For our implementation, each time compartment contains the same number of processes.

In the spatio-temporal pipeline, the ratio between spatial and temporal parallelism can be changed by adjusting the number of processes in a time compartment. Assum-

ing the number of total processes is constant, if the time compartment size is large, then there are few time compartments overall. This leads to lower temporal parallelism, since fewer timesteps are processed concurrently, and higher spatial parallelism, since each timestep will be partitioned into more pieces. On the other hand, if the size of a time compartment is lowered, the total number of time compartments becomes higher. This allows for more timesteps to be processed in parallel, thus temporal parallelism is increased, while lowering the number of pieces each timestep is split into, resulting in less spatial parallelism.

Because it is possible for multiple timesteps to be processed concurrently, one restriction of the spatio-temporal pipeline is that timesteps must be able to be processed independently. Only operations which do not require any communication between timesteps can be used. Examples of such operations include computing the isosurface of each timestep and creating an image for each timestep.

Despite a large number of visualization and analysis algorithms requiring no communication, there are still some operations in which the spatio-temporal pipeline is incompatible. Time-dependent operations which are not associative and require processing through timesteps in a certain order are currently not supported in the spatio-temporal pipeline. This class of operations include pathline advection and Finite-Time Lyapunov Exponent (FTLE) computation.

4. MODELS

As mentioned earlier, large-scale visualization and analysis tasks are usually bottlenecked by the I/O read step. The chosen parallel decomposition approach will determine what the file access pattern is, which greatly affects I/O performance. In order to illuminate how best to configure the spatio-temporal pipeline to get an optimized read pattern, we developed a model of a visualization pipeline. The goal of our model is to compare and determine which parallelization scheme will provide the best performance. A model for the spatio-temporal pipeline is introduced, as well as one for a pipeline using only spatial parallelism.

For modeling purposes, we use the following pipeline:

$$read \rightarrow isosurface \rightarrow write\ isosurface \tag{1}$$

We assume that there is a time-varying dataset stored in the format of each file containing one timestep of a scalar field. Each timestep needs to be loaded from disk. Once the data is loaded into memory, an isosurface is generated. Then the resulting isosurface is written to disk.

4.1 Assumptions

Certain assumptions are made with the models:

- Each file is one timestep containing one scalar field
- Isosurfacing and writes have perfect parallel scaling
- The number of processes allocated per node (*ppn*) is constant
- Each process is run on one core
- In the spatio-temporal pipeline, the total number of processes is evenly divisible by the time compartment size
- In the spatio-temporal pipeline, each time compartment spans the same number of nodes

4.2 Definitions

The following variables are used in the models.

- n is the total number of nodes used
- ppn is the number of processes per node used
- p is the total number of processes, found by $p = n \cdot ppn$
- sf is the size of each file
- pf is the number of processes used to open one file
- nf is the total number of files in the dataset
- mf is the maximum number of files any process will touch
- bw is the bandwidth available to each node
- bwp is the bandwidth available to each process, found by bw/ppn
- tc is the time compartment size

4.3 Spatial Parallelism Model

The spatial parallelism model is based on a decomposition that uses only spatial parallelism, in which all processes are involved in processing each timestep. Therefore, in the spatial parallelism model, $mf = nf$. The total time to compute the pipeline, T_{total}, is found by

$$T_{total} = T_{read} + T_{iso} + T_{write} \tag{2}$$

where T_{read}, T_{iso}, and T_{write} are the time taken in each respective step in the pipeline.

Let us first consider the pipeline steps other than *read* (modeling for the *read* stage is addressed in Section 4.5). These steps are assumed to have perfect linear scaling. Since each file goes through the pipeline, each stage is encountered mf times,

$$T_{iso} = mf \cdot T_{iso_p} \quad , \quad T_{write} = mf \cdot T_{write_p} \tag{3}$$

Where T_{iso_p}, T_{write_p} is the time each respective step takes when p processes are operating in parallel on one timestep. Since these stages are assumed to have perfect linear scaling, the time for each of these stages can be computed with the following equations:

$$T_{iso_p} = \frac{T_{iso_1}}{p} \quad , \quad T_{write_p} = \frac{T_{write_1}}{p} \tag{4}$$

Therefore, T_{total} can be characterized by the equation:

$$T_{total} = T_{read} + mf \cdot \left[\frac{T_{iso_1} + T_{write_1}}{p} \right] \tag{5}$$

4.4 Spatio-Temporal Parallelism Model

In the spatio-temporal pipeline, processes are divided into time compartments. Each time compartment runs in parallel and acts independently of each other. Therefore, the total running time will be the maximum time any time compartment takes. This is equivalent to a time compartment processing mf files. In the spatio-temporal model, mf is found using the equation:

$$mf = \left\lceil \frac{nf}{p \div tc} \right\rceil = \left\lceil \frac{nf \cdot tc}{p} \right\rceil \tag{6}$$

Similar to the spatial parallelism model, the total time is the sum of each step in the pipeline.

$$T_{total} = T_{read} + T_{iso} + T_{write} \tag{7}$$

For all stages except read, the time of each step is

$$T_{iso} = mf \cdot T_{iso_{tc}} \quad , \quad T_{write} = mf \cdot T_{write_{tc}} \tag{8}$$

Similarly to equations 4,

$$T_{iso_{tc}} = \frac{T_{iso_1}}{tc} \quad , \quad T_{write_{tc}} = \frac{T_{write_1}}{tc} \tag{9}$$

Therefore, T_{total} can be written as

$$T_{total} = T_{read} + mf \cdot \left[\frac{T_{iso_1} + T_{write_1}}{tc} \right] \tag{10}$$

How to model T_{read} is discussed in Section 4.5.

4.5 Read Performance Model

We now model the read times of both the spatial parallel model and spatio-temporal model. Note that pf, the number of processes used to open one file, is different for each model. For the spatial parallel model, $pf = p$, while for the spatio-temporal model, $pf = tc$.

First, we start with the read time for one file, T_{read_1}, assuming perfect linear scaling. In this case, the read time is the size of one file divided by the total available bandwidth.

$$T_{read_1} = \frac{sf}{bwp \cdot pf} \tag{11}$$

For both the spatial parallel and spatio-temporal methods, the maximum number of files read by any process is mf, so the total read time for mf files is

$$T_{read_{mf}} = mf \cdot \left[\frac{sf}{bwp \cdot pf} \right] \tag{12}$$

In general, the use of parallelism does not scale perfectly. There is always overhead associated with parallel algorithms, whether it is communication or load imbalance. Since each file is spatially decomposed and read in parallel, we expect there to be overhead for each file read.

$$T_{read_{mf}} = mf \cdot \left[\frac{sf}{bwp \cdot pf} + overhead \right] \tag{13}$$

A parallel file system is a complicated system with many variables and parameters that could affect its performance. In general, a good rule of thumb is that the best I/O performance can be achieved by using contiguous reads. As the number of contiguous reads decrease and the number of file seeks increase, I/O performance will be impaired.

Thus, we base the overhead on the number of file seek operations required to read the file. We assume that the file is written to disk in such a way that spatial coordinates of the x-axis changes fastest, then the y-axis, and finally the z-axis. With this file format, partitions can be read row by row. Therefore the number of seeks can be estimated as the number of rows in a partition, which we denote as ns (number of seeks). Though this is not necessarily the actual number of seeks the disk will perform in a parallel filesystem, we find it is a good estimate. We also believe that as the number of concurrent processes used to read a file grows, the read performance degrades due to increased contention. Because of this, the overhead is also based on the number of processes used to read in a file. Therefore, the final equation for overhead becomes

$$overhead = \alpha \cdot ns + \beta \cdot pf \tag{14}$$

Both α, the time to perform a seek, and β, the amount of contention introduced per process, are free parameters that are based on the hardware characteristics of each machine. The final equation for the read step now becomes

$$T_{read_{mf}} = mf \cdot \left[\frac{sf}{bw \cdot pf} + \alpha \cdot ns + \beta \cdot pf \right] \tag{15}$$

4.6 Analysis of Models

One of the most important inferences we can make from this model is how each method scales as the number of files and processes increase. From the models, we can infer the general performance trend of a weak scaling study (actual results of a weak scaling study are discussed in Section 5). Since it is assumed that the isosurface and write step scale linearly in both methods, the major difference will be how the read step behaves.

As the amount of work and number of processes increase, we expect the spatial parallelism method to incur more read overhead per file. This is because as the number of processes grows, the number of processes used to open a file increases. Also, each individual file will be spatially split into more partitions. This will increase both the ns and pf terms in Equation 14. Thus using spatial parallelism will result in worse file access patterns as the number of processes grows.

For the spatio-temporal pipeline, as weak scaling increases, the time compartment size is kept constant, which means more time compartments are added to process the increased number of files. For example, assume an initial configuration of 4 files and 8 processes with a time compartment size of 4. When doubling to 8 files and 16 processes and a time compartment size of 4, the processes are split into four time compartments, each composed of four processes. In this situation, each file is still read by four processes, so the spatial partitioning remains the same, thus the number of seeks needed remains unchanged. Therefore the resulting overhead value of Equation 14 remains constant. Overall, we expect the spatio-temporal pipeline to scale perfectly in a weak scaling study due to the fact that the read pattern remains unchanged.

This perfect weak scaling of the spatio-temporal pipeline

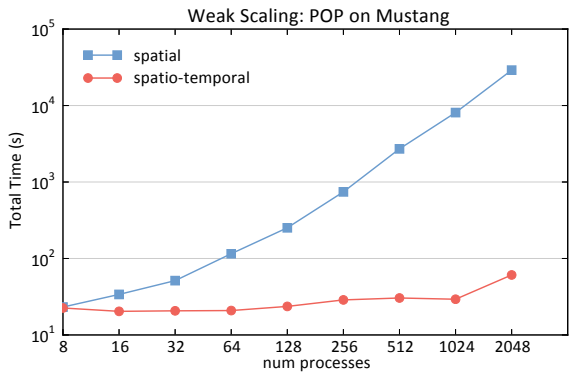


Figure 1: Weak scaling results between the spatio-temporal pipeline and spatial parallelism. Run on Mustang using the POP dataset. The spatio-temporal pipeline with optimized read access patterns scale significantly better than the spatial parallelism method. At 2048 processes, there is a difference of a factor of over 400 between the two methods.

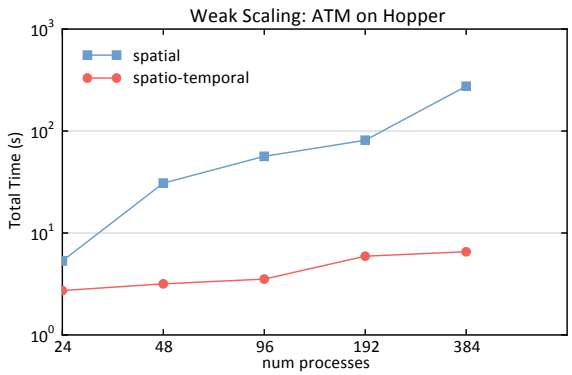


Figure 2: Weak scaling results between the spatio-temporal pipeline and spatial parallelism. Run on Hopper using the ATM dataset. The performance of the spatio-temporal pipeline is orders of magnitude better than the spatial method due to the difference in file access patterns.

implies that reading in multiple files by applying the same read pattern to each file does not create any additional overhead for the filesystem. This assumes that the number of nodes used per file remains fixed so that the amount of bandwidth per file is constant.

5. RESULTS

In order to verify the accuracy of our model, we performed several timings tests. We used two different climate datasets for these tests. The first dataset is output from the Parallel Ocean Program (POP), a simulation of the entire ocean, which we refer to as the *POP* dataset. The data is composed of salinity values, and up to 256 timesteps were used. The spatial resolution of each timestep is 3600 x 2400 x 42. Each file is 1.4 GB, for a total size of roughly 350 GB. The other dataset used is the output from a CAM (community atmospheric model) simulation, which we refer to as *ATM*. Each timestep has spatial resolution of 1152 x 768 x 30, and a total of 16 timesteps were used. For both datasets, files are stored in NetCDF 3 format, and are read using the vtkNetCDFReader class in VTK. The vtkNetCDFReader utilizes the standard NetCDF libraries without any parallel I/O optimizations.

The tests involving the POP dataset were run on *Mustang*, a supercomputer at Los Alamos National Laboratory. Mustang features nodes with dual-socket AMD 12-core MagnyCours and 64 GB of memory. Mustang uses the Panasas filesystem.

All tests with the ATM dataset were run on *Hopper*, a supercomputer at the National Energy Research Scientific Computing Center (NERSC). Hopper contains two 12-core AMD MagnyCours and 32 GB memory per node. Hopper uses the Lustre filesystem.

5.1 Weak Scaling

Weak scaling studies were conducted using both datasets. The POP dataset was run on Mustang. Tests began at one file and 8 processes, and doubled until 256 files and 2048 processes were reached. The number of nodes allocated was equal to the number of files, with 8 cores per node being used. The time compartment size was set to 8 for all tests. With this configuration, each time compartment consisted of 8 processes all located on the same node, and each node held only one time compartment. Each time compartment was responsible for processing one file. Each file was first loaded into memory, then an isosurface was generated, and finally the isosurface was written to disk.

The weak scaling tests using the ATM dataset were similarly configured. All tests were run on Hopper, and the time compartment size was chosen to be 24 for all tests. The number of files began at one and the number of processes started at 24. Subsequent tests doubled these values until 16 files and 384 processes were reached. The number of nodes allocated was equal to the number of files processed, and 24 cores per node were used. Files were processed using the same pipeline as the POP tests described earlier.

Figure 1 shows the results from the POP tests on Mustang, and Figure 2 shows the results from the ATM tests on Hopper. For both tests, the spatio-temporal pipeline displayed significantly better performance and scalability. For the POP tests, at 256 files and 2048 processes, the spatial parallelism method required roughly 29,000 seconds (about 8 hours), while the spatio-temporal pipeline performed the same work using only 60 seconds. This resulted in a speedup factor of over 480. The total time of the spatio-temporal method stayed relatively flat, beginning at 20 seconds, inching up to 30 seconds at 1024 processes, and jumped to 60 seconds at 2048 processes. We believe at 1024 processes, the maximum bandwidth of the system had been reached, thus the doubling of the time. Similar trends are shown for the ATM tests. The spatio-temporal method outscales the spatial method by up to two orders of magnitude. At 394 processes, the spatial method required 274 seconds, while the spatio-temporal method only took 6.5 seconds.

Given the three step pipeline of read, isosurface, and write, our models assumed that the isosurface and write step would scale perfectly. The models also predicted that the read step would increase in time when using only spatial parallelism, and would scale perfectly using the spatio-temporal pipeline due to differences between file access patterns. Figures 3 and

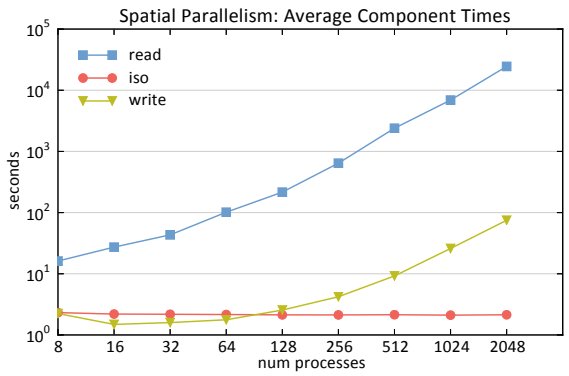


Figure 3: Per component breakdown of results of the weak scaling tests on the POP dataset for spatial parallelism.

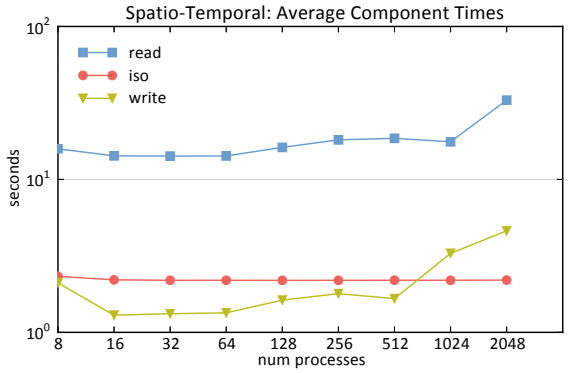


Figure 4: Per component breakdown of results of the weak scaling tests on the POP dataset for the spatio-temporal pipeline.

4 show the actual per component breakdown of the weak scaling results. The spatial parallelism results in Figure 3 show that the isosurface computation step does scale nearly perfectly. The write step begins to increase after 128 processes, but it never consumes more than 1% of the overall running time. As predicted by the model, the read step does steadily increase as the number of processes rise, dominating the overall running time. Thus, the read pattern that resulted by using spatial parallelism greatly impairs I/O read performance. For the spatio-temporal pipeline, Figure 4 shows all three steps scaling well until 2048 processes are used, in which case the read and write times increase. As mentioned earlier, we believe the increase in read times is due to bandwidth limitations, and the rise in write times may also be due to hardware limitations. Up to 1024 processes, the read times remained fairly steady, indicating that the spatio-temporal pipeline used a more optimal file access pattern. Overall, our models have predicted the general trends of both the spatial parallelism and spatio-temporal methods.

Our models not only let us discern the trends of different components, but also can be used to obtain an estimate of the total running time. Many variables, such as size of one file, number of files, and processes per node, are dependent on the run configuration. Other variables, such as *bwp* (the bandwidth available to each process), T_{iso_1}, and T_{write_1}, can be found by performing small timings tests on one node. The number of seeks, ns, can be calculated as the number of rows in each spatial partition. Once all these variables are obtained, they can be plugged into Equation 5 and Equation 10. The two free variables in Equation 15, α and β, are then found by finding the best fit of the modeled times to some actual results on the same machine.

Figure 5 compares the total time estimated from the model and the actual results of the POP weak scaling tests. It was empirically found that $\alpha = 7 \times 10^{-6}$ and $\beta = 1 \times 10^{-3}$ provided the best fit. For the spatial parallelism method, the modeled time tracks fairly close to the actual results. For the spatio-temporal method, the model predicts perfect scaling, so the modeled time is a flat line in the graph. The actual times track the modeled times well, especially at low number of processes. At 2048 processes, the actual time spikes up, but as stated earlier, we believe this is due to bandwidth limitations, which the model does not account for.

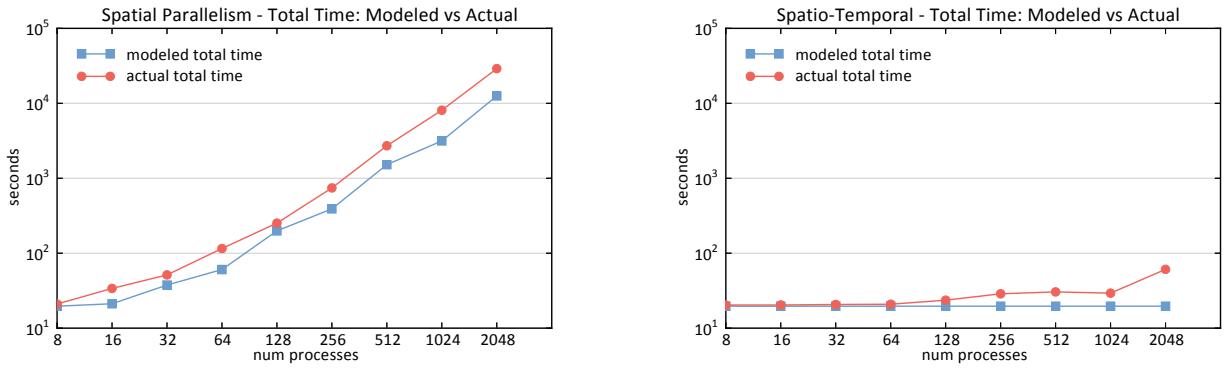
The modeled and actual total time of the ATM weak scaling tests are shown in Figure 6. A best fit was found by using $\alpha = 5 \times 10^{-5}$ and $\beta = 1 \times 10^{-4}$. For the spatial parallelism method, the modeled times track well with the actual times. The greatest difference is at 48 processes, where the model estimate was 30 seconds and the actual time was 11.5 seconds. For the spatio-temporal method, the model always overestimates the total time, but the actual difference is small. Overall, our models predicted the total time of both the POP and ATM tests fairly accurately.

6. DISCUSSION

From the timing tests performed in Section 5, we see that the read pattern that resulted by using the spatio-temporal pipeline resulted in a massive performance increase versus the read pattern induced by using only spatial parallelism. According to our model, the way to reduce I/O read times are to choose read patterns that minimize the number of seeks needed, while also reducing the number of processes reading a file at once. Another result from the model that can be seen in the timing tests is the notion that reading multiple files in parallel using the same read pattern will scale. This scaling can be seen in Figure 1, in which the total time of spatio-temporal pipeline remains fairly flat up to 1024 processes. At 1024 processes, 128 files of size 1.4 GB each are being read in parallel using the same read pattern of 8 processes per file.

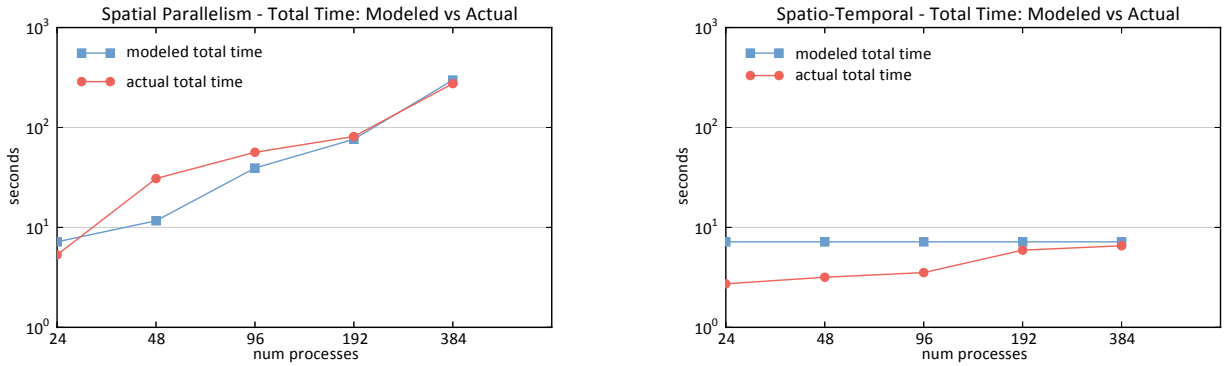
One possible method to reduce I/O read times is to decrease the number of seeks by changing the way spatial partitioning is performed. For example, if a 2D array were stored such that x changed fastest and then y, then partitioning along the y-axis would result in contiguous pieces. Both the spatial parallelism and spatio-temporal method would benefit from more contiguous partitioning. Assuming the data format of one file per timestep, even with a different partitioning scheme, the read patterns from using the spatio-temporal pipeline will probably still have better performance than the file access patterns resulting from the spatial method, since spatial parallelism forces each file to be partitioned into many more pieces. This results in exponentially more seeks when reading data from disk.

The spatio-temporal pipeline introduces one major parameter, the size of a time compartment. This parameter is important because it indirectly controls the file access read pattern. Unfortunately, our model does not explicitly solve



(a) Spatial Parallelism (b) Spatio-Temporal Parallelism

Figure 5: Modeled total times versus actual total times for the spatial parallelism and spatio-temporal weak scaling tests on Mustang for the POP dataset. The model was used with $\alpha = 7 \times 10^{-6}$ and $\beta = 0.001$. Overall, the modeled time duplicates the trends seen in the actual times.



(a) Spatial Parallelism (b) Spatio-Temporal Parallelism

Figure 6: Modeled total times versus actual total times for the spatial parallelism and spatio-temporal weak scaling test on Hopper for the ATM dataset. The model was used with $\alpha = 5 \times 10^{-5}$ and $\beta = 0.0001$. The modeled times are within the same order of magnitude, and track the actual times well at higher process counts.

for this variable, but several guidelines can be suggested. If there is no limit to the number of nodes that can be allocated, then one strategy to obtain the best performance is to first find the optimal number of processes for one file, then scale out by duplicating that configuration to multiple nodes. For example, the configuration of the weak scaling tests on the POP dataset in Section 5 was simply to use 8 processes per node for each file, even though there were 24 cores available per node. According to our model, scaling out by duplicating the run configuration should not increase the total running time of the program. If there is a limit to the number of nodes that can be allocated, then the best strategy would be to try to utilize each node as efficiently as possible. One way to increase efficiency per node is to place multiple time compartments per node, assuming there is enough memory per node. Overall read performance will decrease since a node's I/O bandwidth is now divided over multiple files, but in practice this is offset by the increase in node efficiency. This is due to the fact that reading a file with one node will rarely saturate the network link. For example, changing the previously mentioned configuration for the POP dataset tests to 16 processes and 2 time compartments per node results in each node processing two files at once. This results in an increase of about 20% to total time. The benefit is that only half the number of nodes is needed as before in order to perform the same amount of work. Similar to the earlier guideline, first find the most efficient configuration using one node, and simply duplicate the configuration and scale it out to multiple nodes.

In this paper, we have focused on one very common file format, in which each file is one timestep of one scalar field. Although we have not tested other file formats, our models are general enough that they can be used to estimate the performance of any file format given a certain read pattern. Checking the accuracy of the model and obtaining timing results with a variety of different file formats is left for future work.

7. CONCLUSION

When decomposing a problem into parallel tasks, the read pattern that results from the decomposition is often overlooked. It is critical to understand this effect, since the file access pattern, combined with the format of the stored data, plays a significant role in I/O read performance. In this paper, we have introduced a model which can estimate the I/O read time for a file, given the partitioning of the file. Using this model, coupled with the flexibility of the spatio-temporal pipeline, we were able to generate read patterns

which obtained far greater I/O performance versus spatial parallelism. Several timing tests showed that the optimized file access patterns resulted in a factor of more than 400 speedup. The spatio-temporal pipeline is implemented in ParaView, which is also bundled alongside UV-CDAT [1].

For future work, we plan on studying the performance implications of different file access patterns for with different data formats, such as having multiple timesteps packed into one file.

8. ACKNOWLEDGMENTS

This work has been funded by the UV-CDAT project. The authors would like to thank Matthew Maltrud of Los Alamos National Laboratory and Michael Wehner of Lawrence Berkeley National Laboratory for providing their climate data. This work is sponsored by the Office of Biological and Environmental Research; U.S. Department of Energy. This research used resources of the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

9. ADDITIONAL AUTHORS

10. REFERENCES

[1] UV-CDAT Spatio-Temporal Parallel Processing Tools. `http://uv-cdat.llnl.gov/presentations/PDF/ParaViewSTPWiki.pdf`, 2013.

[2] J. Biddiscombe, B. Geveci, K. Martin, K. Moreland, and D. Thompson. Time dependent processing in a parallel pipeline architecture. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1376–1383, Nov. 2007.

[3] D. Camp, H. Childs, A. Chourasia, C. Garth, and K. I. Joy. Evaluating the benefits of an extended memory hierarchy for parallel streamline algorithms. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 57–64. IEEE, 2011.

[4] H. Childs, D. Pugmire, S. Ahern, B. Whitlock, M. Howison, G. H. Weber, E. W. Bethel, et al. Extreme scaling of production visualization software on diverse architectures. *Computer Graphics and Applications, IEEE*, 30(3):22–31, 2010.

[5] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. E. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 89–96. IEEE, 2011.

[6] W. Kendall, J. Huang, T. Peterka, R. Latham, and R. Ross. Toward a general i/o layer for parallel-visualization applications. *Computer Graphics and Applications, IEEE*, 31(6):6–10, 2011.

[7] C. Michell, J. Ahrens, and J. Wang. Visio: Enabling interactive visualization of ultra-scale, time series data via high-bandwidth distributed i/o systems. pages 1–12. IEEE International Parallel and Distributed Processing Symposium, May 2011.

[8] M. L. Norman and A. Snavely. Accelerating data-intensive science with gordon and dash. In *Proceedings of the 2010 TeraGrid Conference*, page 14. ACM, 2010.

[9] T. Peterka, R. Ross, A. Gyulassy, V. Pascucci, W. Kendall, H.-W. Shen, T.-Y. Lee, and A. Chaudhuri. Scalable parallel building blocks for custom data analysis. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 105–112. IEEE, 2011.

[10] Prabhat, O. Rbel, S. Byna, K. Wu, F. Li, M. Wehner, and W. Bethel. Teca: A parallel toolkit for extreme climate analysis. *Procedia Computer Science*, 9(0):866 – 876, 2012. Proceedings of the International Conference on Computational Science, 2012.

[11] V. Vishwanath, M. Hereld, and M. E. Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 9–14. IEEE, 2011.

[12] B. Whitlock, J. M. Favre, and J. S. Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th Eurographics conference on Parallel Graphics and Visualization*, pages 101–109. Eurographics Association, 2011.

[13] D. Williams, C. Doutriaux, J. Patchett, S. Williams, G. Shipman, R. Miller, C. Steed, H. Krishnan, C. Silva, A. Chaudhary, P. Bremer, D. Pugmire, W. Bethel, H. Childs, M. Prabhat, B. Geveci, A. Bauer, A. Pletzer, J. Poco, T. Ellqvist, E. Santos, G. Potter, B. Smith, T. Maxwell, D. Kindig, and D. Koop. The ultra-scale visualization climate data analysis tools (uv-cdat): Data analysis and visualization for geoscience data. *Computer*, PP(99):1–1, 2013.

[14] M. Woitaszek, J. M. Dennis, and T. R. Sines. Parallel high-resolution climate data analysis using swift. In *Proceedings of the 2011 ACM international workshop on Many task computing on grids and supercomputers*, MTAGS '11, pages 5–14, New York, NY, USA, 2011. ACM.

[15] J. Woodring, S. Mniszewski, C. Brislawn, D. DeMarle, and J. Ahrens. Revisiting wavelet compression for large-scale climate data using jpeg 2000 and ensuring data precision. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 31–38. IEEE, 2011.

[16] H. Yu and K.-L. Ma. A study of i/o methods for parallel visualization of large-scale data. *Parallel Computing*, 31(2):167 – 183, 2005. Parallel Graphics and Visualization.

[17] H. Yu, K.-L. Ma, and J. Welling. A parallel visualization pipeline for terascale earthquake simulations. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, pages 49–, Washington, DC, USA, 2004. IEEE Computer Society.