Adaptive and Efficient Transfer for Online Remote Visualization of Critical Weather Applications

Preeti Malakar[1], Vijay Natarajan[2], and Sathish S. Vadhiyar[2] *

[1] Indian Institute of Technology Kanpur, pmalakar@cse.iitk.ac.in
[2] Indian Institute of Science Bangalore, vijayn@iisc.ac.in,vss@iisc.ac.in

Abstract. Critical weather applications such as cyclone tracking require online visualization simultaneously performed with the simulations so that the scientists can provide real-time guidance to decision makers. However, resource constraints such as slow networks can hinder online remote visualization. In this work, we have developed an adaptive framework for efficient online remote visualization of critical weather applications. We present three algorithms, namely, *most-recent*, *auto-clustering* and *adaptive*, for reducing lag between the simulation and visualization times. Using experiments with different network configurations, we find that the *adaptive* algorithm strikes a good balance in providing reduced lags and visualizing most representative frames, with up to 72% smaller lag than *auto-clustering*, and 37% more representative than *most-recent* for slow networks.

Keywords: weather simulation; remote visualization; adaptivity; representative time step selection.

1 Introduction

High-performance and high-fidelity numerical simulations are important for many scientific and engineering domains such as weather modeling and computational fluid dynamics. Simulations running on thousands of cores take less than a second of execution time per time step [23] and output huge amount of data. While the ability to generate data continues to grow rapidly, the ability to comprehend it encounters great challenges [10, 17]. This is mainly due to the high simulation rates on modern-day processors, as compared to the I/O and network bandwidths. Visualization of simulation output helps in quick understanding of the data, thereby accelerating scientific discovery. However, for critical applications such as hurricane tracking, the simulation output must be simultaneously visualized so that scientists can provide real-time guidance to policy makers. In situ visualization [4, 12] provides some benefits, but has constraints such as physical memory limits, stalling of simulation, supporting only predefined visualizations and requiring modification of simulation code. The scientist may also remotely interact with the output on an analysis or visualization cluster present at supercomputing sites. However, slow speeds in medium/low bandwidth networks between the supercomputing and user's sites can impede interactivity. Another approach is to transfer the simulated data to the user's local site for better interactivity, without stalling the simulation.

Here, we consider this problem of on-the-fly visualization at the user's site such that the user is able to visualize and fully analyze the simulation output locally despite low network bandwidths between the simulation and visualization sites. This alleviates the shortcomings of in situ visualization. Remote visualization of simulation of critical applications, where the visualization is performed at a different location than the site of simulation, also enables geographically distributed scientists to collaboratively analyze the visualization and provide expert opinion on the occurrence of critical events. However, remote visualization may lead to rapid accumulation of data on the storage device (disk, SSD etc.) at the simulation site due to low network bandwidths and limited storage capacity [32]. We developed a framework, InSt, in [20, 21], that adaptively modifies runtime parameters such as simulation speed

* This research was supported in part by Centre for Development of Advanced Computing, Bangalore, India.

and output frequency based on the output of a linear program. Our current work guarantees an upper bound on the *lag* between the time when the simulation produces an output frame (simulation output data for a time step) and the time when the frame is visualized. We use data sieving and data reduction techniques to achieve this. It is important to reduce this lag to enable quick indentification of events from the simulation output and enable the scientists to get an *on-the-fly* view of the simulation.

Simulated frames may accumulate leading to a long queue of pending frames for visualization, and hence increase the number of frames to be sent to the visualization site before the current or recently simulated frame is transferred. The queue length increases with time because the simulation continuously produces output frames. This, in turn, increases the lag between when a frame is produced and when it is visualized assuming frames are sent in the order that they are produced. Our work aims to minimize this lag to enable efficient online visualization. One approach to reduce lag is to sample one or a few frames from the queue. While this approach reduces the lag, it may miss important events. A different approach is to increase the output interval so that no frames are discarded. However, this is equivalent to not examining the simulated data fully and may lead to missing important events. Our approach is to select the most representative frames from the queue and discard the rest. This reduces the queue length while preserving the important events.

Contributions. (1) We have developed three algorithms to reduce the lag between frame creation and visualization – *most-recent*, *auto-clustering* and *adaptive*. An essential criterion is to visualize important events in the simulation. *Most-recent* tries to achieve the best possible lag, *auto-clustering* tries to visualize all important events in the simulation and *adaptive* tries to visualize most of the important events within acceptable lag. (2) These algorithms have been implemented within InSt. InSt adapts to the resource dynamics as a result of executing these algorithms. (3) Using experiments with different network configurations, we find that the adaptive algorithm strikes a good balance between providing reduced lag and visualization of most representative frames, with up to 72% smaller lag when compared to auto-clustering, and 37% more representative than most-recent for slow networks. (4) The experiments show the ability of InSt to adapt to different network bandwidths and yet glean useful information from simulations of critical weather events. Our clustering algorithms are able to deduce the number of distinct temporal phases (clusters) in the data. We demonstrate the efficacy of our algorithms using various metrics.

2 Related Work

On-the-fly visualization: Conventional post processing of simulation output for *offline* visualization is not suitable for critical applications such as weather forecasting, which requires *online* visualization. In situ visualization, where the visualization is done at the same site as simulation, has been extensively studied [1, 4, 6, 8, 9, 12, 18, 30, 32] but has some limitations as follows. The same data structures may be inefficient for both simulation and visualization [18, 30]. The memory requirements (order of TBs [8]) for a coupled simulation and in situ visualization may exceed the limited physical memory per node on supercomputers. Deciding visualization/analysis parameters a priori may be challenging for critical applications, as used in prior work [3, 15]. ParaView Catalyst [3, 9] requires the visualization pipeline configurations to be predecided before processing the current simulation time step output. Adaptable I/O system (ADIOS) [5, 15] use data staging and in situ data transformations on the output data. Libsim [32] reads simulation data from physical memory when required by the VisIt [7] server. SENSEI [4] can use additional infrastructures to transfer data between simulation and visualization/analysis. Simulation code needs to be modified in the above approaches, which may stall the simulation while data is transferred to visualization. They also do not consider remote visualization scenarios and poor network bandwidths between simulation and visualization sites. In this work, we consider remote visualization where the scientist/user visualizes locally (at the remote site) and has greater control over the data. We enable the user to perform full visual analysis locally and on-the-fly without stalling the simulation. This is helpful because analysis scenarios are often not known a priori, especially for

critical applications. While existing frameworks support zoom-in features where it is left to the user control based on the requirements, our framework performs data selection adaptively based on the lag.

One can also generate the visualized results (images) at the simulation site and send them to the user's site. Cinema [26] allows scientists to decide the parameters for images a priori. Images are typically smaller than compressed raw data, and hence results in faster data transfers. However, in many cases the users may require raw data for detailed analysis. If the user wants to modify the visualization pipeline, the request has to be sent to the simulation site, which in turn will increase the interaction time of the scientist. Also, the user cannot explore and interact with the entire mesh, going back and forth in time. In our model, we transfer the simulation output, which gives full flexibility to the scientist to interact with the full mesh. We also perform remote visualization of the data sent by simulation over networks which may have low bandwidths. Though the transfer time can vary from seconds to minutes depending on the amount of data and the network bandwidth, our approach of transferring simulation output to user's site for visualization is a viable option for critical and petascale applications due to more interactivity at the user's site and expensive compute hours at the supercomputing site.

Selection of representative frames: Keyframe selection has been studied for summarizing video [14, 16]. However, it is not directly applicable to simulation output selection for online visualization because we want to select the most representative frames considering the current lag. In real-time video transmission, each frame may be encoded using different bit rate depending on the perceptual quality required for that frame [24]. In our case, capturing key events is more preferable unlike video transmission, where continuity is important to avoid perceptual disturbance in on-demand video transmission [25], unlike our case. Shen et al. [29] exploit data coherency between consecutive simulation time steps. Differential information is used to compute the locations of pixels that need updating at each time step. This is useful when a large percentage of values remain constant between consecutive time steps, as also reported by them. We found that there are more than 80% changed elements between successive output steps in our application. Hence sending differential information will lead to diminishing returns in our case. Wang et al. [31] derive an importance measure for each spatial block in the joint feature-temporal space of the data based on conditional entropy formulation using multidimensional histograms. Based on clustering of the importance curves of all the volume blocks, they suggest effective visualization techniques to reveal important aspects of time-varying data. Their histogram calculation takes 19 hours for a 960×660×360 volume for 222 time steps with block size of 48×33×18. Such a long time for selection of important frames from a running simulation is clearly unsuitable for efficient online visualization. Patro et al. [27] measure saliency in molecular dynamics (MD) simulation output. Their keyframe selection method considers atom positions to determine saliency of the atoms in a time step and then aggregate information to find saliency of the time step. Their saliency function is specific to MD simulations where interesting and purposeful molecular conformational changes occur only over larger time scales.

3 Adaptive Integrated Framework

We use our adaptive steering framework, InSt (see Figure 1, details in [20,21]), that performs automatic tuning and user-driven steering to enable simultaneous simulation and online remote visualization. The *simulation* process is the weather application that simulates weather events across time steps and outputs weather data to available storage (such as non-volatile memory, burst buffers, SSD, disk etc.). InSt transfers data from the simulation to the visualization site. There is an *adaptive frame sender* at the simulation site and *frame receiver* at the visualization site. The remote *visualization* process continuously visualizes the simulation output. InSt adapts to resource constraints and an *application manager* determines final execution parameters for smooth and continuous simulation and visualization of critical applications in resource-constrained environments. However, it cannot guarantee an upper bound on the *lag* between the time when the simulation produces an output frame and the time when the frame is visualized. When the network bandwidth is high, the frames are transferred quicker. In

case of low network bandwidths, the frames take longer time to be transferred and hence the number of frames simulated during that time is higher. This leads to lag accumulation.

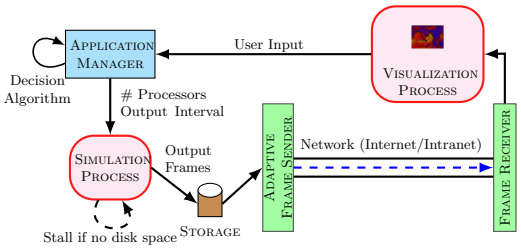


Fig. 1: InSt: Adaptive Integrated Steering Framework

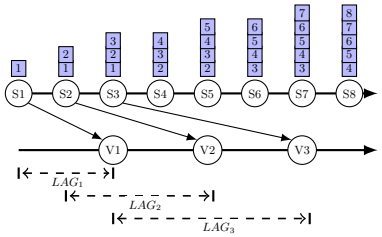


Fig. 2: Simulation and visualization progress

Figure 2 shows increasing lag between the simulation and visualization. The two horizontal lines show simulation and visualization progress. Si is the simulation time for the i^{th} frame and Vi is the visualization time for the same. LAG_i shows the difference between Vi and Si, i.e. the time difference between the visualization and simulation of the i^{th} frame. Note that when the 1^{st} frame reaches the visualization site at $V1$, the 2^{nd} and 3^{rd} frames are already produced and are waiting to be sent. When the 2^{nd} frame reaches the visualization site at $V2$, the 3^{rd}, 4^{th} and 5^{th} frames are queued at $S5$. Though the i^{th} frame is produced at Si, it can only be sent after the previously-queued frames are transferred. Therefore the queue size continues to increase as shown by the numbered rectangles in Figure 2. This is true for all kinds of intermediate storage. Since the number of frames waiting at the simulation site increases, the time between when a frame is produced and when it is visualized also increases. For example, the 8^{th} frame will have to wait in queue until the previous 7 frames are sent. The transfer times of the queued frames add to the lag for the 8^{th} frame. This leads to cumulative addition of lag for the later frames. Hence LAG_i increases as illustrated here for the 1^{st}, 2^{nd} and 3^{rd} frames. InSt invokes a frame selection algorithm to send a subset of frames to the visualization site. We modified InSt's *frame sender* to incorporate the adaptivity, as discussed next.

4 Reduction of Simulation-Visualization Lag

The scenario shown in Figure 2 can be better or worse depending on the network bandwidth between the simulation site and the visualization site. Since the simulation output data size is large, therefore such data transfers will be bandwidth-limited. While latency will be a factor when considering small data, we consider only network bandwidth in our work that considers large data. For low-bandwidth networks, the lag accumulation will be high. So, an online visualization framework should adapt to the network bandwidth and minimize the lag. We have developed strategies in InSt that adapt to the network bandwidth and the length of the queue of pending frames i.e., the frames that are yet to be sent to the visualization site from the parallel simulation site.

4.1 Requirements for Online Visualization

The lag between $S3$ and $V3$ is more than between $S1$ and $V1$ (from Figure 2), i.e. the lag for the frames produced later is more than the lag for the earlier frames. This increasing lag is mainly due to two reasons – (1) Sending all frames and (2) More frames are output by the simulations while a frame is being transferred. A simple strategy to decrease lag is to increase the output interval i.e. to not produce excess frames if the network bandwidth is low. However, this may result in missing important events between the two frames. Since the purpose of visualization is to identify important events, this strategy is not desirable. The lag for successive frames increases with increasing queue of pending frames. Thus, we decrease the queue length by dropping some frames from the queue to decrease the lag. In our work, we choose a subset of frames from the queue and discard the rest so that

the queue size decreases, which in turn will reduce the lag. Our framework adapts to different network bandwidths to reduce the lag. The higher the network bandwidth, the lesser the number of frames that will be dropped by InSt. The criterion to drop frames must adhere to either of the two conflicting goals: Case 1: The sent frames contain useful information - The goal is to send good quality frames. The quality of the frames may be based on the amount of non-redundant information contained in them. These are the most representative frames, i.e. the frames that are distinct from each other and represent their immediate temporal neighborhood well. However this will not give the best possible lag because there may be many important frames in the queue. Case 2: Minimal lag is maintained - In this case, the goal is to always maintain the best possible lag irrespective of whether all important information is visualized or not. Thus we need to either compromise the quality of visualized frames or the simulation-visualization lag. We extended InSt to dynamically decide which pending frames would be sent, as detailed next.

4.2 Strategies for Selection of Time steps to Reduce the Lag

We have incorporated the following three frame selection algorithms within InSt.

Most-recent (mr) This simple strategy selects the frame that is most recently generated by the simulation to send to the remote visualization site. Let t_cur be the current time when a frame is chosen, t_gen be the time when the frame was generated by the simulation, and t_tran be the time for transferring the chosen frame to the visualization site. Then, the most-recently generated frame results in minimal value of $(t\_cur - t\_gen) + t\_tran$ among all the frames in the queue, since t_gen for the most recent frame is the highest. Thus *mr* aims to reduce the lag between the visualization and simulation time of a frame to the minimal value. However, note that the most recent frame may not be the most representative frame of all the frames in the queue. Alternatively, one can select a representative frame from the queue of pending frames. This algorithm takes constant time.

Fig. 3: Auto-clustering Strategy

Auto-clustering (auto) This algorithm reduces the lag as well as sends only useful information to the visualization site. Sending all the frames can result in large simulation-visualization lag (discussed in Section 4.1). *Auto* selects some representative frames from the queue of pending frames so that useful information is not lost and important frames are retained. We decide the importance of a frame based on how well that frame represents the other frames in its temporal neighborhood. This is because it is important to visualize the significant temporal phases in the output. In the first part of this algorithm, we examine the current queue of pending frames and form temporal non-overlapping clusters as shown in Figure 3. The different colors represent different clusters. These clusters represent phases in the queue of pending frames. The phases are determined by comparing the root mean square distance of the values of a varying field between two successive frames. Given two sets of N points, P_1 and P_2, the root mean square distance and the normalized root mean square distance are given by Equations (1) and (2) respectively, where x is the value.

$$RMSD(P_1, P_2) = \sqrt{\frac{\sum_{i=1}^{N}(x_{1,i} - x_{2,i})^2}{N}} \quad (1) \qquad NRMSD = \frac{RMSD}{x_{max} - x_{min}} \quad (2)$$

The algorithm determines the number of clusters based on the root mean square distance. After forming the clusters, a representative frame from each cluster is chosen such that it has the least standard deviation among the frames in that cluster. These are colored black in Figure 3. The pseudocode is

Input: The set of pending frames $\mathcal{F}$

1 **foreach** $i \in \mathcal{F}$ **do**
2 | Find $RMS(f_{i-1}, f_i)$ using Equation (2);
3 | $rms[f_i] \leftarrow RMS(f_{i-1}, f_i)$;
4 **end**
5 $avg\_rms \leftarrow$ average of $rms[f_i] \; \forall \; i \in \mathcal{F}$;
6 $k \leftarrow 0$; /* k is number of clusters */
7 **foreach** $i \in \mathcal{F}$ **do**
8 | **if** *(avg_rms – rms*$[f_i] \geq$ *threshold)* **then**
9 | | $k \leftarrow k + 1$;
10 | **end**
11 **end**
/* Let $\mathcal{C}(\mathcal{G}_1), \mathcal{C}(\mathcal{G}_2), ..., \mathcal{C}(\mathcal{G}_k)$ be the frames that represent the centers of k clusters. */
/* Initially, the clusters centers are equally spaced. Refine the cluster centres. */
12 **repeat**
13 | **foreach** $j \in \mathcal{F}$ **do**
14 | | $\mathcal{G}_p \leftarrow \underset{i=\{left,right\}}{\text{argmin}} (RMS(f_j, \mathcal{C}(\mathcal{G}_i)))$;
15 | | add j to members($\mathcal{G}_p$);
16 | **end**
17 | **foreach** $i \in \mathcal{G}$ **do**
18 | | $\mathcal{C}(\mathcal{G}_i) \leftarrow \underset{i \in members(\mathcal{G}_i)}{\text{argmin}}$ standard deviation(i);
19 | **end**
20 **until** *there is no change in the cluster centres*;
21 **foreach** $i \in \mathcal{G}$ **do**
22 | $\mathcal{R}_i \leftarrow \mathcal{C}(\mathcal{G}_i)$;
23 **end**

Output: The set of representative frames $\mathcal{R}$

Algorithm 1: Auto-clustering Algorithm

shown in Algorithm 1. The algorithm takes as input the set of pending frames $\mathcal{F} = \{f_1, f_2, ..., f_n\}$, that are queued at the simulation site and outputs the set of representative frames $\mathcal{R} = \{r_1, r_2, ..., r_k\}$, $\mathcal{R} \subseteq \mathcal{F}$, to be sent. The number of clusters k is determined in lines 1–11. We calculate the normalized root mean square distance (NRMSD) of pressure variable between every two consecutive frames using Equation (2) and find the standard deviation. The value of pressure decreases over time, thereby increasing the function range. We use NRMSD in order to avoid biasing the RMSD by higher function range. The number of clusters is determined by the number of frames having a high standard deviation. We have chosen a threshold of 0.4 standard deviation above the mean to define large distances of frames from the previous frames. From initial observations, we found that 0.4 standard deviation provides a balance between very few and very high number of clusters. Thus, the algorithm uses the principle that if f_i is distinctly different from f_{i+1}, it may imply a change of phase.

Once the number of clusters k is determined, we find the cluster centres using an iterative method similar to the well-known k-means [19] clustering algorithm. Unlike the traditional k-means, we aim to find a temporal clustering, which means that the clusters are sequenced according to the temporal order. The reason for this is to capture distinct temporal phases among the queued frames. Each cluster has a common boundary with each of its neighboring cluster to its right and left sides (Figure 3). Initially, we place the cluster centres at equal distance from each other. In every iteration, each frame is assigned to the closest cluster centre among the two centres to its left and right (lines 13–16 of Algorithm 1). After assigning the frames to one of the cluster centres to its left or right, a new cluster centre is determined for each cluster based on the standard deviation of root mean square distance. In each of

the clusters, the one that has the least standard deviation is selected as the new cluster centre (lines 17–19). This is continued until there is no change in cluster centres, which are our representative frames. The space requirements for this algorithm are modest because only the data points and centroids are stored. Specifically, the storage required is $O(n + k)$, where n is the number of points and k is the number of clusters. The time required is $O(I * n)$, where I is the number of iterations required for convergence. We have found empirically that this algorithm converges fast and I is very small, therefore this algorithm is linear in the number of data points. Although auto-clustering does not guarantee minimal lag, it selects representative frames from the queue of pending frames to retain the frames that contain significant information and important phases in the parallel simulation are visualized.

Adaptive (adaptive) This hybrid strategy combines the characteristics of the *mr* and the *auto* because it is important to reduce the lag as well as to visualize the important phases of the simulation. *Adaptive* gives utmost importance to visualizing the frames as soon as they are produced by the simulation process. The user/scientist can specify an upper bound LAG_UB to limit the simulation-visualization lag. As discussed in Section 4.1, one way to reduce lag is to drop frames. But dropping many frames may result in too much loss of information. Another approach to reduce lag without losing too much information is to reduce the size of each frame. Since we employ frame compression as a size reduction technique in all our strategies, we consider another kind of size reduction in this adaptive strategy. The specific size reduction technique is to remove less important information from a frame. Scientific data produced by simulations has different sets of parameter values, and these sets can be prioritized into different levels of importance based on the specific needs of the scientists. For example, weather data has different sets of variables such as pressure, temperature, wind velocities, humidity, precipitation, etc. For the critical weather application of cyclone tracking, pressure is the most important variable. A cyclone is characterized by continuous drop in pressure and high wind velocity at the centre of the cyclone. So we form different levels of information by retaining different sets – (1) Level 0: All variables, (2) Level 1: Pressure, Wind Velocity, Temperature and (3) Level 2: Pressure. Thus, we can adaptively reduce the frame size to different levels by retaining different sets of most useful data in the frame. Since the data size affects the data transmission time, it will decrease if we reduce the size of each frame by sending the most important information in the frame. This is the form of data reduction that we adopt in our work. Note that we do not reduce the high spatial resolution of output data in order to preserve high fidelity. Scaling down frames would hamper the data fidelity, especially for extreme events.

Input: The set of pending frames $\mathcal{F}$ and the set $\mathcal{L}$ of different levels of information in descending order of amount of information content

`/* Invoke auto-clustering algorithm to get the set of representative frames` $\mathcal{R}'$ `*/`

1 $\mathcal{R}'$ = auto-clustering($\mathcal{F}$)

`/* Adaptively choose an appropriate level for the chosen frames */`

2 **foreach** *representative frame* $i \in \mathcal{R}'$ **do**
3 **foreach** *level* $j \in \mathcal{L}$ **do**
4 $curr\_frame \leftarrow j^{th}$ level of information in i;
5 $curr\_transfer\_time \leftarrow$ time to transfer $curr\_frame$;
6 **if** $(curr\_transfer\_time \leq LAG\_UB)$ **then**
7 add i to $\mathcal{R}$;
8 break;
9 **end**
10 **end**
11 **end**

Output: The set of representative frames $\mathcal{R}$

Algorithm 2: Adaptive Algorithm

It may not be always possible to send the full simulation output to the visualization site within the lag limit, so this algorithm tries to send as much information as possible for visualization. At first, the adaptive algorithm invokes the auto-clustering algorithm explained in Section 4.2. For each of the representative frames output by auto-clustering, the adaptive algorithm checks if the full frame can be sent without violating the lag limit LAG_UB, i.e. it checks whether the difference in the times between when the frame will reach the visualization site and when it was produced by the simulation process will be less than LAG_UB. If it cannot send the full frame without violating the lag limit, then it checks whether it can send the frame with the next level of reduced information content such that the lag is less than LAG_UB. There can be multiple such levels of reduced information content depending on the amount of information in the simulation output. With each level of reduced information in a frame, the time to send the frame also decreases since the time to transfer data is directly proportional to its size. If even the lowest level of reduced frame content cannot be sent, then the adaptive algorithm discards the frame and considers the next representative frame. The pseudocode for this is shown in Algorithm 2.

This technique ensures that the lag for the visualized frames is always less than LAG_UB. When the algorithm decides to send a frame with partial data, the time to transfer that frame is less which implies that the number of pending frames accumulated in the queue within that time is fewer. Hence the rate at which the queue length increases is lower when reduced frames are sent. When fewer frames are pending, then in the next iteration, the algorithm will most likely select a full frame for visualization within LAG_UB. Hence the *adaptive* algorithm is able to adapt to network conditions and current queue size. It adaptively decides whether to send or not and how much information to send. We elaborate this using experimental results in Section 5. The time complexity of *adaptive* is similar to that of *auto*.

4.3 Implementation

InSt invokes the frame selection algorithm when the frames are in transit from the simulation to the visualization site, hence this ensures that the time required for the frame selection does not increase the simulation-visualization time. We remove the frames from the simulation site once they are transferred to the visualization site. The time required to cluster is proportional to the number of pending frames. For example, in the high-bandwidth case, the maximum queue length is 3 when frame selection algorithm is used. The transfer time of a full frame at 18 km resolution is around 1 minute and the frame selection algorithm runs in less than 0.3 seconds for clustering 3 frames. However, in certain cases like slow network bandwidths, where the queue size can be very large, the frame selection algorithm execution time may exceed the frame transfer time. In those cases, the frame selection algorithm considers a subset of the pending frames so that its execution time does not surpass the transfer time. The limitation of this approach is that considering a subset of frames may result in choosing different representative frames. However, we have found experimentally that this simple modification maintains the quality of the data reasonably well as has been shown in Section 5.

5 Experiments and Results

In this section we present the details of the weather application used for simulation and remote visualization, the resource configuration for the experiments and the results of our frame selection algorithms.

Weather Model and Cyclone Tracking We used a weather forecast model, WRF (Weather Research and Forecasting Model) [22] as the simulation process for simulating weather events. The modeled region of forecast is called a *domain* in WRF. There is one parent domain which can have child domains, called *nests*, to perform finer level simulations in specific regions of interest. We used our framework, InSt for tracking a tropical cyclone, *Aila* [2], in the Indian region. The cyclone was formed on May 23, 2009 about 400 kms south of Kolkata, India and dissipated on May 26, 2009 in

the Darjeeling hills. We used the *nesting* feature of WRF to track the lowest pressure region or eye of the cyclone and perform finer level simulations in the region of interest. The nesting ratio i.e. the ratio of the nest resolution to that of the parent domain, was set to 1:3. We performed simulations for an area of approximately 32×10^6 sq. km. from 60°E - 120°E and 10°S - 40°N over a period of 3 days. InSt forms the nest dynamically based on the lowest pressure value in the domain and monitors the nest movement in the parent domain along the eye of the cyclone. A nest is spawned at the location of the lowest pressure when the pressure drops below 995 hPa. We use a configuration file that specifies different simulation resolutions for different pressure gradients or cyclone intensities. This can be specified by meterologists who typically use coarser resolutions for the initial stages of cyclone formation and finer resolutions when the cyclone intensifies. As and when the cyclone intensifies i.e. the pressure decreases, InSt refines the nest resolution multiple times to obtain a better simulation result from the model. The finest nest resolution was 4 km. InSt adapts the output frequency based on the LP [20].

Framework Details The modifications to the WRF weather application for our work are minimal. We modified WRF to include monitoring of lowest pressure in the nest domain or in the parent domain when there is no nest. WRF is restarted whenever pressure drops below the threshold values specified by the user. When the application configuration file specifies the number of processors and output interval that are different from the current configuration, the simulation process is rescheduled on a different number of processors.

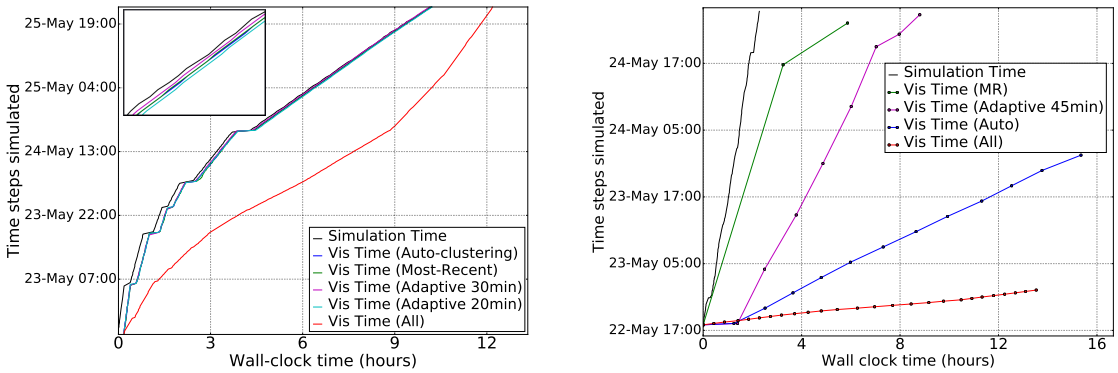
The output file sizes of the parent domain varies from 300 MB to 4.1 GB per time step, depending on the resolution and the level chosen by our adaptive approach. Finer resolution and full data (level 0) results in maximum output size. For faster I/O we used WRF's split NetCDF approach, where each processor outputs its own data. For example, simulation of 12 km resolution running on 288 processes results in output of 3 MB per process per time step. The split approach is beneficial, especially for low bandwidth, low latency networks for faster data transfer from the simulation site. It also significantly reduces the I/O time per time step by up to 90% over the default approach of generating output to a single large NetCDF file. We have developed a utility to merge these split NetCDF files at the visualization site. Our plug-in for VisIt [7] enables *directly reading* the WRF NetCDF output files, eliminating the cost of post-processing before data analysis. We also customized VisIt to automatically render as and when these WRF NetCDF files are merged after arriving at the visualization site. The simulation output has been visualized using GPU-accelerated volume rendering, vector plots employing oriented glyphs, pseudocolor and contour plots of the VisIt visualization tool. For steering, we have developed a GUI inside VisIt using Qt.

System Configuration We ran WRF simulations on two sites with different remote visualization settings – *high-bandwidth* and *low-bandwidth*. For the high-bandwidth configuration, simulations were executed on the *fire* cluster in Indian Institute of Science. *Fire* is a 20-node dual-core AMD Opteron 2218 cluster, each node has 4 GB RAM. It has 250 GB hard disk and is connected by Gigabit Ethernet. For the low-bandwidth configuration, simulations were done on the Cray XT5 supercomputer, *kraken* [11], at the National Institute for Computational Sciences (NICS) of Oak Ridge National Laboratory and University of Tennessee, Knoxville, USA. *Kraken* has two six-core AMD Opteron processors (Istanbul) and 16 GB RAM per node and connected by Cray SeaStar2+ router. We used a maximum of 48 cores on *fire* and 288 cores on *kraken* for simulation. The average simulation-visualization bandwidth was 56 Mbps and 1.1 Mbps for *fire* and *kraken* respectively. We used the average observed bandwidth (obtained by the time taken to transmit 1 GB message) between the simulation and visualization sites. For all experiments, visualization was performed on a graphics workstation in Indian Institute of Science (IISc) with a Intel(R) Pentium(R) 4 CPU 3.40 GHz and an NVIDIA graphics card GeForce 7800 GTX.

Results Next, we present experimental results of using the frame selection algorithms for simulation-visualization lag reduction by InSt. As explained in Section 4.2, for the *adaptive* algorithm, the user/scientist can specify an upper bound on the simulation-visualization lag. We experimented with

upper bounds of 20, 30 and 45 minutes for simulation-visualization lag. The frame selection algorithms are sequential in the current work, and are executed on a single processor at the simulation site. We measure the performance improvement over the original *all* algorithm in terms of simulation-visualization lag. We also compare the frame selection algorithms in terms of reproducibility of information with respect to the original set of frames. This is to demonstrate that the resulting visualization does not hinder the quality of data for scientific analysis.

Simulation-visualization Lag Figure 4a shows lag between the simulation and visualization times



(a) High-bandwidth network configuration. (b) Low-bandwidth network configuration.

Fig. 4: Simulation-visualization lag. Black curve shows simulation times.

for various frame selection algorithms in high-bandwidth configuration. The x-axis shows the wall clock time for simulation and visualization. The y-axis shows the simulation time steps. The red "all" curve shows the visualization times for the default policy of sending *all* frames generated at the simulation site. We find the curves to be overlapping because for a given time step on y-axis, the difference in wall clock times between simulation (black) and visualization time is very small for our algorithms (as shown by the inset image on the top left corner that zooms part of the curves). This is because the frame selection algorithms are able to prune the frames to be sent and hence can reduce the *lag accumulation* (see §4). Selectively sending pending frames and high network bandwidth minimize the queue length and reduce the lag. Figure 4b illustrates simulation-visualization lag in low-bandwidth configuration. Note that *most-recent* performs the best in terms of lag. The *adaptive* algorithm considerably improves the simulation-visualization lag (reduces the lag by ∼86%) because it sends the representative frames only if it can meet the lag bound. However, we observe time difference in hours for a bound of 45 minutes because the network bandwidth in this configuration varies, which affects the transfer times. Note that this transfer happens over the internet between NICS and IISc, and thus it is difficult to predict the current available network bandwidth. We will look into incorporating a better (internet) network bandwidth estimator in future. The lag for *auto-clustering* is better than the *all* but worse than the *adaptive* because the *auto* algorithm sends all the representative frames.

RMS distance We compute the root mean square (RMS) distance between successive frames to measure distortion. Minimizing mean squared error leads to better perceptual quality [24]. The RMS distance between successive frames is a measure of continuity and captures the variation between successive frames. If a frame selection algorithm has the same variation as the original *all* algorithm, then the frame selection algorithm closely follows the original algorithm. The average RMS distance for all, auto-clustering, most recent and adaptive with 20 minutes lag bound for high-bandwidth configuration are 0.0039, 0.0045, 0.0045 and 0.0044 respectively. We compute the RMS distance with respect to the variable perturbation pressure P. We do not observe a significant variation in the average RMS distance between successive frames for different frame selection algorithms. The average RMS distance for the frame selection algorithms do not differ much from the original *all* strategy. This suggests that there is no major information lost even if frames were dropped at the simulation site. The

average RMS distance for all, auto-clustering, most recent and adaptive with 45 minutes lag bound for low-bandwidth configuration are 0.001, 0.007, 0.018 and 0.009 respectively. The RMS distance for frames sent by *most-recent* is quite high due to the huge gap between the successive frames. The average RMS distance for the *adaptive* algorithm is higher than *auto-clustering* because the latter has lesser output interval between two successive frames. Though the *auto-clustering* algorithm has more simulation-visualization lag, it provides better continuity between the frames. The *adaptive* algorithm strikes a good balance in providing reduced simulation-visualization lag and choosing the most representative frames.

Histogram We examine the frequency distribution of the data which gives the spread or variability in the data values from the *all* and the frame selection algorithms. Similar frequency distributions imply similarity in the frequencies of the range of data values output by the algorithms. The frequency distribution can be obtained from the histogram of the data distribution in the frames selected by the frame selection algorithms. The histogram for *all* is shown in Figure 5 for the variable perturbation pressure. Most of the pressure values for all the time steps lies in the bin number 150, which corresponds to the perturbation pressure range of -92 Pa to -82 Pa. We show the histogram similarity between the *all* and the frame selection algorithms by calculating the volume between the histograms. The volume enclosed between the histograms of *auto*, *mr*, *adaptive* with lag bounds of 30 and 20 minutes and the *all* algorithm are 12.16, 14.00, 16.12 and 9.95 respectively for the high-bandwidth configuration. Lesser the volume, more similar are the histograms. The minimum volume is for the *adaptive* algorithm with lag bound of 20 minutes, followed by *auto-clustering*. The *adaptive* algorithm with lag bound of 20 minutes sends some frames with reduced information which leads to reduced transfer times and hence more frames can be sent. Therefore it is able to perform better than *auto-clustering* which always sends full frames. Both these algorithms perform better than the *most-recent* because the most recently produced frame may not be the most representative frame in the queue. The histogram for the *adaptive* algorithm with lag bound of 30 minutes is the most dissimilar to the histogram for *all* algorithm. This is because the lag limit of 30 minutes allows full frames to be transferred initially because of higher lag limit in comparison to the lag of 20 minutes. However, later, the framework is unable to send frames even with reduced information because the full frames incur high transfer times, which increases the pending frame count. Therefore, during the simulation, sometimes the frames in the front of the queue cannot be sent in order to maintain the lag bound. This leads to discarding many representative frames and thus the *adaptive* algorithm with lag limit of 30 minutes performs worse than the others.

The volume between the histogram for *all* and the histograms for *auto*, *mr* and *adaptive (45 minutes lag bound)* are 58.38, 161.12 and 101.52 respectively for the low-bandwidth configuration. The *auto-clustering* volume is the lowest, i.e. it is the most similar to *all*. This is because *auto* sends all the representative frames unlike *most-recent* and *adaptive*. Though *mr* has the lowest lag (see Figure 4b), it is most dissimilar to *all*. Hence sending the most recent frame in the queue may not be the best strategy to reduce simulation-visualization lag with respect to quality of the frames sent. *Adaptive* performs better than the *mr* but its volume difference from the *all* strategy is larger than for *auto* because the *adaptive* never sends any frame if it is unable to meet the specified lag bound. The higher the lag bound, more the number of frames that can be sent by the *adaptive* algorithm. Higher lag bound also improves the information content of the frames sent.

Nest position changes In WRF, a moving nest captures the movement of the cyclone and the nest centre is placed at the eye of the cyclone. An important feature of cyclone tracking is to track the eye of the cyclone. Hence the nest movement in WRF is important because it follows the movement of the eye of the cyclone. However, the eye of the cyclone, and therefore the nest, does not move at every time step. Though the frame selection algorithms drop frames in order to reduce the lag, they ideally should capture the frames in which the nest position changes. Hence we compare the algorithms in terms of how well the frames chosen by the algorithms capture the nest movement. The number of nest position changes captured by *all*, *auto*, *mr*, *adaptive* with lag bound of 30 minutes and *adaptive* with lag bound of 20 minutes are 114, 102, 97, 94 and 106 respectively for high-bandwidth configuration.

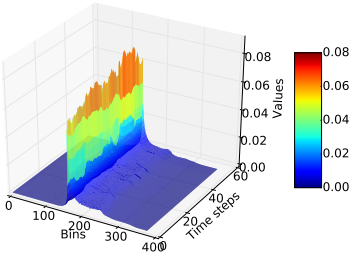


Fig. 5: Histogram for *all* algorithm for *high-bandwidth* configuration

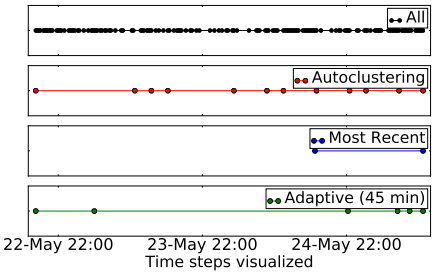
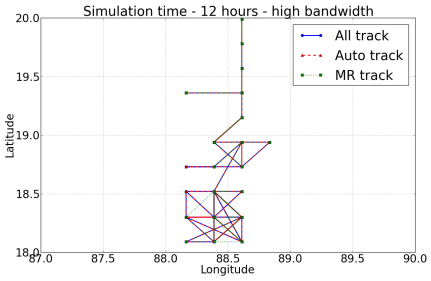


Fig. 6: Nest position changes for *low-bandwidth* configuration

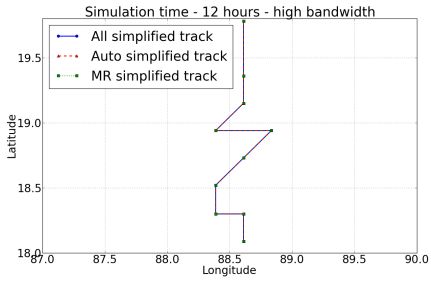
This shows that the *adaptive* with lag bound of 20 minutes and the *auto-clustering* algorithms are able to capture most of these nest movements.

Figure 6 shows the frames in which the nest position changes from the position in the previous frame for low-bandwidth configuration. The dots in the graphs depict the frames corresponding to changes in nest positions. These are shown for those frames that are chosen by the frame selection algorithms. The number of nest position changes captured by *all*, *auto-clustering*, *most-recent* and *adaptive* with lag bound of 45 minutes are 148, 12, 2 and 6 respectively. When compared to the high-bandwidth configuration results, the low-bandwidth configuration leads to very small number of nest position changes captured by the algorithms due to the slow network. The *auto* and *adaptive* strategies lead to better distribution of nest position changes than the *most-recent* that captures the nest position changes only at the later part of the application progress. This is because *mr* selects the most recent frame in the queue without any consideration about the representativeness of the chosen frame. So it is possible that the chosen frame does not have changes in the nest position. *Auto* captures the nest position changes best at the expense of increased simulation-visualization lag. When compared to *adaptive*, *auto* exhibits a steady-state behavior of sending the most representative frames, and hence has a higher chance of sending frames with nest position changes.

Cyclone track The track of the cyclone shows its path from its formation to its dissipation. The



(a) Cyclone tracks for *all*, *auto* and *mr* from the simulation.



(b) Simplified cyclone tracks for *all*, *auto* and *mr* algorithms.

Fig. 7: Comparison of Aila tracks for *all*, *auto* and *mr* algorithms.

cyclone track is essentially the path of the eye of the cyclone, and is an important outcome of the high performance simulation. Hence, we also compare the cyclone tracks generated by the *all*, *auto* and *mr* frame selection algorithms. Figure 7 shows the track of the lowest pressure point in the simulation from 24^{th} May 2009 06:00 hours to 24^{th} May 2009 18:00 hours. Here, we represent the track as a polyline [28]. Figure 7a shows the tracks when *all* the frames were sent and when *auto-clustering* and *most-recent* algorithms were used to send frames from simulation to visualization. It is challenging to get accurate simulated track [13]. Figure 7b shows the simplified tracks for *all*, *auto* and *mr* for high-bandwidth configuration after removing intersecting line segments and zero-length line segments.

We also removed small angles, small line segments and loops to simplify the track. The simplified tracks of all three are coincident. This shows that the output from our algorithms show similar simulated tracks as *all*.

6 Conclusions and Future Work

We presented an adaptive framework for high-resolution simulation and online remote visualization of critical weather applications such as cyclone tracking. The goal of this work is to help scientists who run simulations at high-performance computing sites and would like to visualize results on-the-fly at their local sites and perform instantaneous analysis. In some cases, it may not be preferable to do in situ visualization at the simulation sites due to low-bandwidth networks and ability to explore and interact better locally.

We described *most-recent*, *auto-clustering* and *adaptive* algorithms for frame selection at the simulation site in order to reduce the simulation-visualization lag. We showed that the *most-recent* performs the best in terms of lag-reduction but it cannot ensure that representative frames are selected. The *adaptive* algorithm helps both in reducing the lag as well as improving the information content of the visualized frames. *Auto-clustering* performs well in terms of sending representative frames from the simulation site and reduces lag as compared to *all*. *Auto* and *adaptive* are able to form temporal clusters (i.e. identify distinct phases) from the data without any user guidance regarding the number of clusters. For high-bandwidth networks, *auto* and *adaptive* result in small delays. For low-bandwidth networks, the maximum simulation-visualization lag that can be tolerated may be decided by the user. *Adaptive* can modify the frame content and frequency to meet the user's requirements. Using experiments with different network configurations, we find that the *adaptive* algorithm balances well between providing reduced lags and visualizing most representative frames, with up to 72% smaller lag than *auto-clustering*, and 37% more representative than *most-recent* for slow networks.

In future, we plan to investigate research challenges related to multiple simultaneous simulations and visualizations. Our framework can be extended to simultaneously support various visualization requirements from different geographically distributed users, thereby enabling collaborative analysis and visualization. We also plan to apply our techniques for remote visualization of other applications such as remote online medical image analysis and, viewing and steering of astronomy data.

References

1. J. Ahrens et al. An image-based approach to extreme scale in situ visualization and analysis. In *SC14: Proc. Intl. Conf. High Performance Computing, Networking, Storage and Analysis*, pages 424–434, 2014.
2. Cyclone Aila. `http://en.wikipedia.org/wiki/Cyclone_Aila`.
3. U. Ayachit et al. 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.
4. U. Ayachit et al. Performance Analysis, Design Considerations, and Applications of Extreme-Scale In Situ Infrastructures. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 921–932, 2016.
5. D. A. Boyuka et al. Transparent in situ data transformations in adios. In *Proc. IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 256–266, May 2014.
6. F. Chen et al. Enabling In Situ Pre- and Post-processing for Exascale Hemodynamic Simulations - A Co-design Study with the Sparse Geometry Lattice-Boltzmann Code HemeLB. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 662–668, 2012.
7. H. R. Childs et al. A Contract Based System For Large Data Visualization. In *IEEE Visualization*, 2005.
8. D. Ellsworth et al. Concurrent Visualization in a Production Supercomputing Environment. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):997–1004, 2006.
9. N. Fabian et al. The ParaView Coprocessing Library: A scalable, general purpose in situ visualization library. In *IEEE Symposium on Large Data Analysis and Visualization*, 2011.
10. C. D. Hansen and C. R. Johnson. *The Visualization Handbook*. Academic Press, 2005.

11. Kraken Cray XT5, NICS, Tennessee. `http://www.nics.tennessee.edu/computing-resources/kraken`.
12. J. Kress et al. Loosely Coupled In Situ Visualization: A Perspective on Why It's Here to Stay. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, 2015.
13. A. Kumar, J. Done, J. Dudhia, and D. Niyogi. Simulations of Cyclone Sidr in the Bay of Bengal with a high-resolution model: sensitivity to large-scale boundary forcing. *Meteorology and Atmospheric Physics*, 114(3–4):123–137, 2011.
14. T. Liu, H.-J. Zhang, and F. Qi. A Novel Video Key-frame-extraction Algorithm based on Perceived Motion Energy Model. *IEEE Transactions on Circuits and Systems for Video Technology*, 2003.
15. Q. Liu et al. Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience*, 26:1453–1473, 2014.
16. J. Luo, C. Papin, and K. Costello. Towards Extracting Semantically Meaningful Key Frames From Personal Video Clips: From Humans to Computers. In *IEEE Transactions on Circuits and Systems for Video Technology*, pages 289–301, 2009.
17. K. L. Ma. In Situ Visualization at Extreme Scale: Challenges and Opportunities. *IEEE Computer Graphics and Applications*, 29(6):14–19, 2009.
18. K.-L. Ma, C. Wang, H. Yu, and A. Tikhonova. In Situ Processing and Visualization for Ultrascale Simulations. *Journal of Physics (Proceedings of SciDAC 2007 Conference)*, 78, 2007.
19. J. B. MacQueen. Some Methods for Classification and Analysis of MultiVariate Observations. In *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.
20. P. Malakar, V. Natarajan, and S. Vadhiyar. An Adaptive Framework for Simulation and Online Remote Visualization of Critical Climate Applications in Resource-constrained Environments. In *Proceedings of the 2010 ACM/IEEE conference on Supercomputing*, 2010.
21. P. Malakar, V. Natarajan, and S. Vadhiyar. InSt: An Integrated Steering Framework for Critical Weather Applications. In *Proceedings of the International Conference on Computational Science*, 2011.
22. J. Michalakes et al. The Weather Research and Forecast Model: Software Architecture and Performance. In *Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, 2004.
23. J. Michalakes et al. WRF Nature Run. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, page 59, 2007.
24. A. Ortega and K. Ramchandran. Rate-distortion methods for Image and Video Compression. *Signal Processing Magazine, IEEE*, 15(6):23 –50, 1998.
25. T. Ozcelebi, A. Tekalp, and M. Civanlar. Delay-Distortion Optimization for Content-Adaptive Video Streaming. *Multimedia, IEEE Transactions on*, 9(4):826 –836, June 2007.
26. P. OLeary et al. Cinema image-based in situ analysis and visualization of MPAS-ocean simulations. *Parallel Computing*, 55:43–48, 2016.
27. R. Patro, C. Y. Ip, and A. Varshney. Saliency Guided Summarization of Molecular Dynamics Simulations. In *Scientific Visualization: Advanced Concepts*, pages 321–335, 2010.
28. K. N. Scheitlin, V. Mesev, and J. B. Elsner. Polyline averaging using distance surfaces: A spatial hurricane climatology. *Computers & Geosciences*, 52(0):126–131, 2013.
29. H.-W. Shen and C. R. Johnson. Differential Volume Rendering: A Fast Volume Visualization Technique for Flow Animation. In *Proceedings of the conference on Visualization '94*, pages 180–187, 1994.
30. T. Tu et al. From Mesh Generation to Scientific Visualization: an End-to-End Approach to Parallel Supercomputing. In *SC '06: Proc. of the ACM/IEEE conference on Supercomputing*, 2006.
31. C. Wang, H. Yu, and K.-L. Ma. Importance-Driven Time-Varying Data Visualization. In *IEEE Transactions on Visualization and Computer Graphics*, 2008.
32. B. Whitlock et al. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Eurographics Symposium on Parallel Graphics and Visualization*, 2011.