Comparing the Efficiency of In Situ Visualization Paradigms at Scale

James Kress[1,2], Matthew Larsen[3], Jong Choi[1], Mark Kim[1], Matthew Wolf[1], Norbert Podhorszki[1], Scott Klasky[1], Hank Childs[2], and David Pugmire[1]

[1] Oak Ridge National Laboratory, Oak Ridge TN 37830, USA
{kressjm,choij,kimmb,wolfmd,pnorbert,klasky,pugmire}@ornl.gov
[2] University of Oregon, Eugene OR 97403, USA
hank@uoregon.edu
[3] Lawrence Livermore National Laboratory, Livermore CA 94550, USA
larsen30@llnl.gov

Abstract. This work compares the two major paradigms for doing in situ visualization: in-line, where the simulation and visualization share the same resources, and in-transit, where simulation and visualization are given dedicated resources. Our runs vary many parameters, including simulation cycle time, visualization frequency, and dedicated resources, to study how tradeoffs change over configuration. In particular, we consider simulations as large as 1,024 nodes (16,384 cores) and dedicated visualization resources with as many as 512 nodes (8,192 cores). We draw conclusions about when each paradigm is superior, such as in-line being superior when the simulation cycle time is very fast. Surprisingly, we also find that in-transit can minimize the total resources consumed for some configurations, since it can cause the visualization routines to require fewer overall resources when they run at lower concurrency. For example, one of our scenarios finds that allocating 25% more resources for visualization allows the simulation to run 61% faster than its in-line comparator. Finally, we explore various models for quantifying the cost for each paradigm, and consider transition points when one paradigm is superior to the other. Our contributions inform design decisions for simulation scientists when performing in situ visualization.

1 Introduction

The processing paradigm for visualizing simulation data has traditionally been post hoc processing. In this mode, the simulation writes data to disk, and, at a later time, a visualization program will read this data and perform desired analyses and visualizations. However, this mode is severely handicapped on today's HPC systems, as computational capabilities are increasingly outpacing I/O capabilities [3, 11].

To alleviate this pressure on the I/O systems, in situ processing methods [8] are now being used for analysis and visualization of simulation data while it is still in memory, i.e., before reaching disk. In situ methods are varied, but in broad terms can be placed in two categories: in-line in situ and in-transit in situ.

In the in-line in situ paradigm (sometimes also referred to as tightly coupled in situ), the simulation and the visualization will directly share the same set of resources. In this paradigm, generally speaking, the simulation will compute a specified number of cycle iterations and then pause while the visualizations are performed. Once the visualizations have been computed, the simulation will continue on to the next time step. In the in-transit in situ paradigm (sometimes also referred to as loosely coupled in situ), the simulation and visualization use separate resources. In this paradigm, the simulation will compute a specified number of cycle iterations and then transfer the simulation data over the network to the dedicated visualization resources. Once this transfer is completed, the simulation runs concurrently to the visualization tasks being performed.

Both paradigms have been applied successfully for real HPC applications. In a typical application, any publication evaluating in situ processing will generally include only anecdotal information that shows the simulation benefited from in situ processing, perhaps including comparisons with the post hoc paradigm. However, there has been substantially less research dedicated to how these two paradigms directly compare. As a result, it is difficult to understand which paradigm to use for a particular situation. A thorough comparison requires consideration of many different axes [17], e.g., execution time, cost, ease of integration, fault tolerance, etc. In this work, we focus our scope to execution time and cost. The goal of this work is to understand the performance of these two paradigms for a number of simulation configurations.

Our hypothesis entering this work was that both paradigms (in-transit and in-line) are useful, i.e., some workloads favor one paradigm with respect to execution time and cost, and other workloads favor the other, and a major contribution of this work is confirmation for that hypothesis. In particular, we have found that visualization workloads are different than the more general analysis workloads that have been studied previously, and so the best approaches for visualization differ. Specifically, we find that the rendering operation inherent to visualization has parallel coordination costs, which makes in-transit more competitive in comparison to analysis-centric workloads, since in-transit will often run with fewer nodes and so the coordination costs are reduced. Further contributions of this work include additional analysis of when to choose which processing paradigm and why, with respect both to time to solution and to resources used.

2 Related Work

While the constraints of the I/O systems in current HPC systems have made in situ visualization an important topic, many of the central ideas go back to the early years of computing. Bauer et al. [8] provide a detailed survey of the history of in situ visualization.

Over the years, a number of infrastructures for doing in situ visualization have been widely used. SCIRun [23] is a problem solving environment that allowed for in situ visualization and steering of computations. Cactus [16] provides a framework to assist in building simulation codes with plug-ins that can per-

form tasks such as in situ visualization. LibSim [26] is a library that allows simulations to use the full set of features of the VisIt [10] visualization tool for in situ exploration, extraction, analysis and visualization. ParaView Catalyst [6] offers a similar in situ functionality for the ParaView [4] visualization tool. ADIOS [19] is an I/O middleware library that exposes both in-line and in-transit paradigms to a simulation through a POSIX-like API. Its in-transit capabilities are provided by a number of different data transport methods, including DataSpaces [13], DIMES [27], and FlexPath [12]. Damaris/Viz [14] provides both in-line and in-transit visualization using the Damaris I/O middleware. Ascent [18] is a fly-weight in situ infrastructure that supports both distributed-memory and shared-memory parallelism. SENSEI [5] is a generic data interface that allows transparent use of the LibSim, Catalyst, and ADIOS in situ frameworks.

Several large studies have been done on using in situ methods in HPC simulations. Bennett et al. [9] use both in-line and in-transit techniques for analysis and visualization of a turbulent combustion code. Ayachit et al. [7] performed a study of the overheads associate with using the generic SENSEI data interface to perform in situ visualization using both in-line and in-transit methods. These and other studies are focused on the particular methods chosen for in situ visualization. They do not do a comparison between in-line and in-transit methods, nor discuss the tradeoffs associated with each.

Adhinarayanan et al. [2] on the other hand look at characterizing in-line in situ vs. post-hoc processing from the energy usage point of view. Their goal was to see if in-line in situ was more energy efficient for a simulation vs. post-hoc processing. Similarly, Gamell et al. [15] look at energy usage vs. performance for an in-line in situ analytics pipeline, and explore ways of reducing the energy usage with in situ processing. Rodero et al. [25] use the same concept and expand it to look at different configurations of simulation and visualization nodes to reduce energy usage.

Our work takes a different view than any of these works. First, we focus specifically on in situ visualization pipelines, which tend to have different communication and computation scaling curves than a full scale simulation. Second, we focus specifically on in-line in situ vs. in-transit in situ, and look specifically at visualization frequency, resource requirements, and how different combinations of all of these factors impact the bottom line of simulation scientists in terms of compute time used for visualization pipelines.

The closest comparator to our own work work comes from Oldfield et al. [22]. Their work also considered in-line and in-transit in situ. However, their work was primarily focused on analysis use cases, where our work is focused on scientific visualization use cases. This difference is essential, because scientific visualization use cases involve rendering which requires parallel image compositing. This image compositing can become a bottleneck at large scale. This bottleneck is particularly relevant to this problem because the in-line approach operates with higher concurrency and thus suffers a bigger delay, while the in-transit approach performs image compositing at lower concurrence and thus less delay. As a result, our findings differ than those of Oldfield et al., specifically that in transit is

superior for a much higher percentage of workloads than Olfield's analysis-based study. We also consider a wider array of factors, including varying the number in-transit resources, the simulation cycle time, and perform additional evaluation by including a total cost model.

3 Experimental Overview

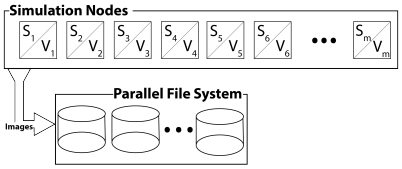
In evaluating the characteristics of the in-transit and in-line in situ paradigms, we hypothesize that there are particular configurations for the simulation and analysis such that one paradigm outperforms the other. This includes both performance metrics under consideration in this work: execution time and resource cost. One intuition is that in-transit in situ has a chance to execute the analysis algorithms using fewer overall cycles, since it will use lower node counts and suffer less busywaiting from bottlenecks. This potential benefit, however, must offset the cost of allocating extra nodes, as well as transferring data from the simulation. To test our hypothesis, we designed a set of experiments to study the behaviors of both paradigms.

For this study, we use CloverLeaf3D [1, 20], a hydrodynamics proxy-application that solves the compressible Euler equations. Cloverleaf3D spatially decomposes the data uniformly across distributed memory processes, where each process computes a spatial subset of the problem domain; it does not overlap communication with computation. To couple CloverLeaf3D with both in-transit and in-line in situ, we leveraged the existing integration with Ascent [18]. For in-transit visualization, Ascent's link to the Adaptable I/O System (ADIOS) [19] was used to transport data, and then the distributed memory component of VTK-m [21] was used to perform the visualization tasks. For in-line visualization, Ascent applied the distributed memory component of VTK-m directly. As a result, the same visualization code was being called on the same data sets in both settings, with the only differences being (1) whether Ascent used ADIOS to transport the data and (2) the number of nodes dedicated to visualization.

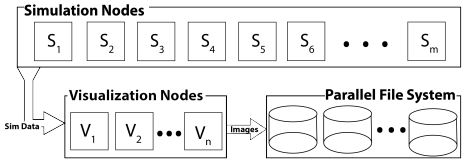
Visualization Tasks: The visualization tasks performed were isocontouring and parallel rendering. These tasks were chosen because they are widely used in scientific visualization, and parallel rendering is a communication-heavy algorithm that allowed testing of the performance bottleneck hypothesis. After each simulation time step, isocontours of the energy variable were computed at values of 33% and 67% between the minimum and maximum value for each time step. Since energy is a cell-centered quantity in CloverLeaf3D, the variable had to be re-centered, i.e., cell values surrounding each point were averaged. After the isocontours are computed, the geometry is rendered to an image using a parallel rendering algorithm. As we are operating in a distributed memory environment, each MPI process locally rendered the data it contained, then all of the locally rendered images were composited using radix-k.

In-line Visualization Setup: In-line visualization is accomplished via Ascent. Ascent's main visualization capability is effectively as a distributed memory

version of VTK-m. The visualization is described through a set of actions. Ascent combines these actions into a data flow graph, then executes the graph. The in-line setup is illustrated in Figure 1a. Again, for this case, the simulation and visualization share the same resources.



(a) Representation of the in-line visualization used as part of this study. With this mode, the simulation and visualization alternate in execution, sharing the same resources.



(b) Representation of the in-transit visualization used as part of this study. With this mode, the simulation and visualization operate asynchronously, and each have their own dedicated resources.

Fig. 1: Comparison of the two workflow types used in this study.

In-transit Visualization Setup: In-transit visualization is accomplished via Ascent's link with ADIOS. ADIOS is only used in the in-transit case because the data needs to be moved off node before visualization can take place, whereas visualization is done in place in the in-line case. ADIOS supports memory-to-memory data transports between processes or applications. I.e., it supports transporting data in a memory space of one application to the memory space of another. For this study, we used the DIMES data transport method. In the DIMES data transport method, the writing process transports the data asynchronously over the remote direct memory access network (RDMA) to the reading process. Additionally, DIMES requires the use of metadata servers to hold indexing information for the reading processes. The in-transit setup is shown in Figure 1b. Here, a dedicated set of resources are used for the visualization. After the simulation has computed a time step, the data are transferred over the network to the visualization resources where visualization is performed asynchronously. Comparisons between these two methods are presented in Section 4. Because the two paradigms use different numbers of resources, we use two evaluation metrics to make a fair comparison. The first, time to solution, is discussed in Section 4.1, and the second, total cost, is discussed in Section 4.2.

Experiments: There are several different variations of each in situ paradigm. Examples include whether the same memory space is used for in-line in situ, or how proximate the visualization resources are for in-transit in situ. In this study we focus on the most common variation for each. We also consider configurations that directly affect in situ performance, such as simulation cycle time, visualization frequency, and resources dedicated to in-transit in situ. We evaluate the implications of these configurations both in terms of total time to run the

simulation, and in terms of total resources used. We also explore the scalability of in situ visualization in both paradigms, and the implications of visualization performed at various levels of concurrency.

The experiments were designed to build a better understanding of the performance of the in-line and in-transit in situ paradigms. To aid in the analysis of this experiment, we ran a number of different in situ configurations:

- **Sim only:** Baseline simulation time with no visualization
- **In-line:** Simulation time with in-line visualization
- **Alloc(12%):** In-transit visualization allocated an additional 12% of simulation resources
- **Alloc(25%):** In-transit visualization allocated an additional 25% of simulation resources
- **Alloc(50%):** In-transit visualization allocated an additional 50% of simulation resources

For the in-transit paradigm, predetermined percentages of simulation resources for visualization were selected. These percentages, listed above, were selected based off of a rule of thumb in the visualization community where 10% of resources are traditionally devoted to visualization. We used that rule as a starting point and used two additional higher allocations to explore a range of options for simulation scientists. This also allows enough range to study the right ratio of simulation and visualization resource allocations. We also initially considered in-transit allocations that were below 10%, but due to the memory limitations on Titan (32 GB per node), the visualization nodes ran out of memory. Because of this, we omitted these experiments from our study. In the in-line case, visualization had access to all of the simulations resources. Finally, for all tests, we ran each one of these configurations in a weak scaling study with concurrency ranging between 128 and 16,384 processes, with 128^3 cells per process (268M cells to 34.4B cells).

Because CloverLeaf3d is a mini-app using a simplified physics model, the simulation has a relatively fast cycle time. This fast cycle time is representative for some types of simulations, but we also wanted to study the implications with simulations that have longer cycle times. To simulate these longer cycle times, we configured CloverLeaf3D to pause after each cycle completes, using a sleep command. This command was placed after the simulation computation, and before any visualization calls were made. To ensure no simulation communication was done asynchronously during the sleep call or visualization routines, the simulation tasks were synchronized before entering sleep. The three cases used were:

- **Delay(0):** simulation ran with no sleep command.
- **Delay(10):** a 10 second sleep was called after each simulation step.
- **Delay(20):** a 20 second sleep was called after each simulation step.

Conceptually, longer cycle times benefit in-transit visualization. Figure 2 demonstrates how visualization latency is hidden in in-transit vs. in-line visualization. After the data have been transferred to the visualization resources,

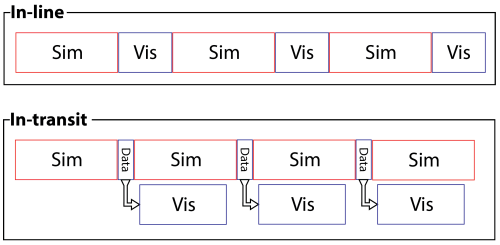


Fig. 2: Gantt chart showing how the simulation and visualization progress over time (from left to right) with both in-line and in-transit in situ. In this notional example, the data transfer for in-transit is faster than the visualization step for in-line, meaning the in-transit simulation can advance more quickly (four cycles versus three).

the simulation and visualization proceed in parallel allowing in-transit to hide the latency of the visualization. In-line visualization cannot take advantage of latency hiding.

Lastly, in the bulk of the experiments we fixed the visualization frequency to once every time step. This is a common setup in codes that evolve quickly, where skipping timesteps could cause events to be missed. By having very frequent visualization, we can see an upper bound for how visualization will impact the simulation. To contrast these results we did a small study with a visualization frequency of once every three simulation cycles to see how the simulation is impacted. This scenario, where visualizations are performed every n cycles, is also common, and we wanted to understand how the frequency of visualization compared in terms of time and cost.

Hardware: The experiments in this study were performed on the Titan supercomputer deployed at the Oak Ridge Leadership Compute Facility (OLCF) at Oak Ridge National Laboratory. Titan is a Cray XK7, and is the current production supercomputer in use at the OLCF. It contains 18,688 compute nodes and has a peak performance of 27 petaflops. Because the mini-app we used for our study runs on CPUs only, we restricted this study to simulations and visualizations run entirely on the CPU. This also simplifies the analysis as we are not concerned with data movement within the node (from GPU to network).

Launch Configuration: The configuration for each experiment performed is shown in Table 1. Because CloverLeaf3D is not an OpenMP code, the in-line in situ and the simulation only configurations were launched with 16 ranks per node. The in-transit configurations used 4 ranks per visualization node and 4 OpenMP threads to process data blocks in parallel. Therefore, in-transit and in-line both used 16 cores per node. In the in-transit configuration, each rank will be assigned multiple blocks. Additionally, the in-transit configuration required the use of dedicated staging nodes to gather the metadata from the simulation in

Table 1: Resource configuration for each of the tests performed in our scaling study.

Test Configuration	**Sim Processes**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**
	Tot. Data Cells	648^3	816^3	1024^3	1296^3	1632^3	2048^3	2592^3	3264^3
In-line	**Total Nodes**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
In-transit *Alloc*(12%)	Vis Nodes	1	2	4	8	16	32	54	128
	Staging Nodes	1	2	2	4	4	8	8	16
	Total Nodes	**10**	**20**	**38**	**76**	**148**	**296**	**584**	**1168**
In-transit *Alloc*(25%)	Vis Nodes	2	4	8	16	32	64	128	256
	Staging Nodes	1	2	2	4	4	8	8	16
	Total Nodes	**11**	**22**	**42**	**84**	**164**	**328**	**648**	**1296**
In-transit *Alloc*(50%)	Vis Nodes	4	8	16	32	64	128	256	512
	Staging Nodes	1	2	2	4	4	8	8	16
	Total Nodes	**13**	**26**	**50**	**100**	**196**	**392**	**776**	**1552**

order to perform RDMA memory transfers from the simulation resource to the visualization resource. These additional resources are accounted for in Table 1.

4 Results

The objective of our experiments was to understand the performance of in situ visualization using both in-line and in-transit paradigms and explore the hypotheses presented in Section 3. Our results focus on time to solution (Section 4.1), total cost (Section 4.2), and performance and load balancing of visualization algorithms (Section 4.3).

4.1 Time to Solution

Figure 3 shows the total runtime for each study configuration. There are several insights that can be drawn from Figure 3. First, the in-line visualization operations in our study are subject to poor performance as concurrency increases (see Section 4.3 for a discussion on scalability). Second, the simulation cycle has a large impact on how many resources are required for in-transit visualization to outperform in-line visualization. In *Delay*(0), where simulation cycle times are very quick, the *Alloc*(50%) configuration is required for the in-transit resources to keep up with the simulation. As the simulation cycle time increases in *Delay*(10) and *Delay*(20), fewer visualization resources are required to out perform. In the case of *Delay*(20), the performance of all the in-transit configurations are nearly identical.

The times for each configuration are a result of the work required to perform in situ visualization, and are different for each paradigm. In Figure 3, the added time for in situ visualization is indicated by the gap between the "Sim Only" line, and the in situ configuration lines. For example, in the *Delay*(0) case with 16,384 processes, the sim-only time was 561 seconds, while the in-line time

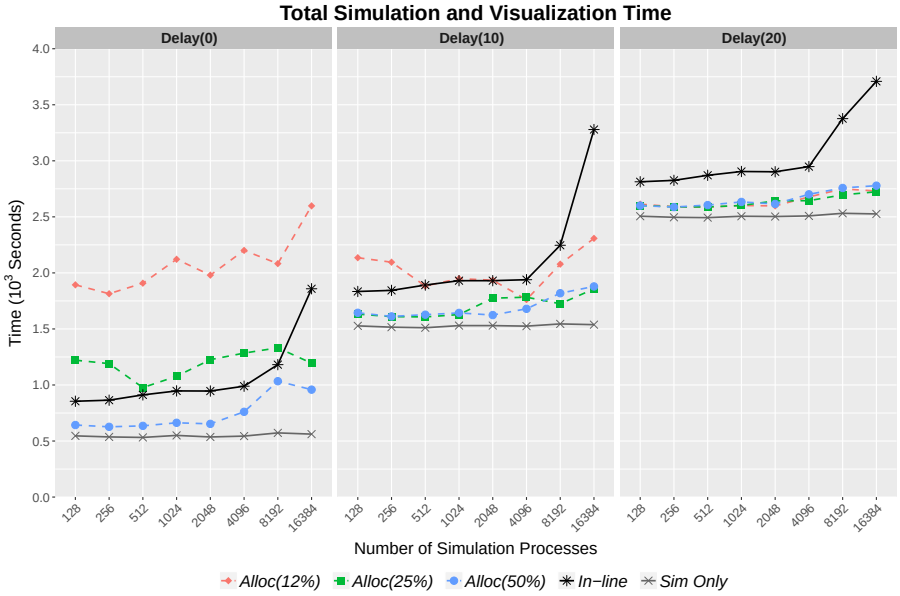


Fig. 3: Total execution time for the three simulation configurations ($Delay(0)$, $Delay(10)$, $Delay(20)$) using in-line visualization, and three configurations of in-transit visualization ($Alloc(12\%)$, $Alloc(25\%)$, $Alloc(50\%)$).

was 1,858 seconds, and increase of 1,297 seconds to do visualization. This gap is a result of the simulation stalling for the visualization. For in-line visualization, the simulation will stall until the visualization operations are complete, at which point the simulation will continue with the next time step. For in-transit visualization, the simulation is stalled while the data are transferred to the visualization resources. Once the transfer is complete, the simulation will continue with the next time step, and the visualization will be performed concurrently on the dedicated resources (see Figure 2). In-transit visualization is also subject to second type of stall, which can occur when the time to complete the visualization tasks exceeds the cycle time of the simulation. We permit such stalls to occur in our experiments. An alternative would have been to only begin visualization tasks if resources are available (i.e., drop time slices of data). We felt permitting stalls showed more interesting behavior, as the result from dropping time slices is approximately the same as increasing the simulation cycle time — which we cover in other experiments.

For in-line visualization, the simulation stall is the direct cost of the visualization operations. The amount of simulation stall increases with the level of concurrency, and is a result of a drop in the scalability of the visualization operations. This effect can be seen at higher level of concurrency, and will be discussed later in Section 4.3.

For in-transit visualization, the simulation stalling in Figure 3 is more complicated. In these cases, the simulation is stalled by the data transfer time, and

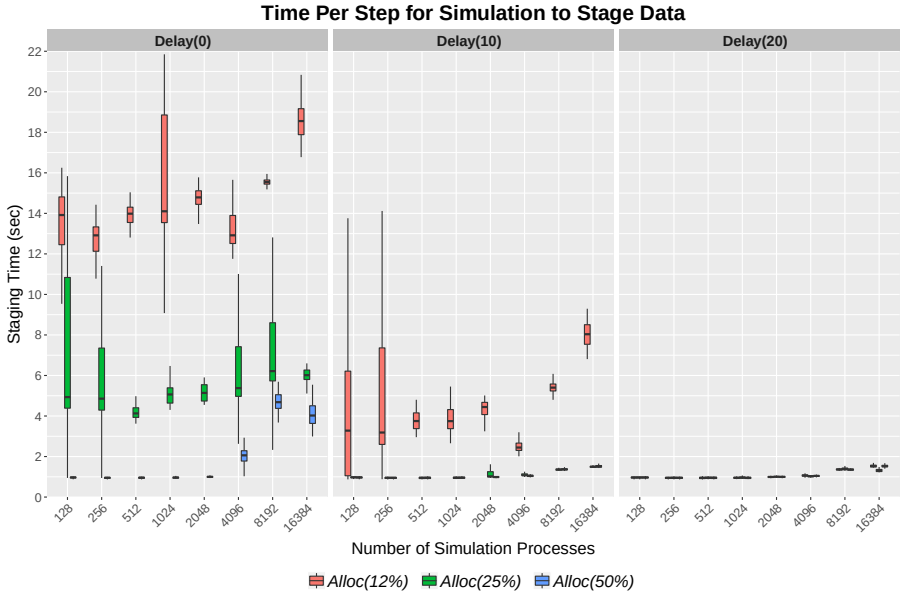


Fig. 4: Boxplot of the time it took to write data from the simulation to the staging resource at each step during the simulation. Results are shown for the three simulation configurations of in-transit visualization. This chart demonstrates the extent to which in-transit visualization slowed down the simulation. The lower the staging time, the less time it took the simulation to write data and continue on to the next cycle. Note that the majority of the time to stage data is due to the simulation being stalled while waiting for the visualization resources to free up (Delay(0) case), and in general staging the data is a quick operation (Delay(20) case).

in some cases, while waiting for the visualization processes to catch up. For example, in the in-transit $Delay(0)$ $Alloc(25\%)$ case in Figure 3, there is a rise in time between 1024 and 8192 processes. Figure 4 shows the range of data transfer times over all time steps in the simulation. Larger boxes in Figure 4 indicates longer data transfer times which corresponds to the stalling described above. There is a correspondence between the stall times in Figure 4, and the total times in Figure 3. For example, looking at concurrency of 1024, 2048, 4096 and 8192 for $Delay(0)$ $Alloc(25\%)$ cases in Figures 3 and 4 shows the increase in time is due to stalling. As the simulation cycle time increases in $Delay(10)$ and $Delay(20)$, and the visualization has more time to keep up with the simulation, the stalling decreases.

Figure 5 is a metric that quantifies the impact to the simulation by the visualization. Given a fixed time allocation of 500 seconds, the graphs show how many simulation time steps can be completed with each configuration. The case where no visualization is performed is the high water mark for each graph.

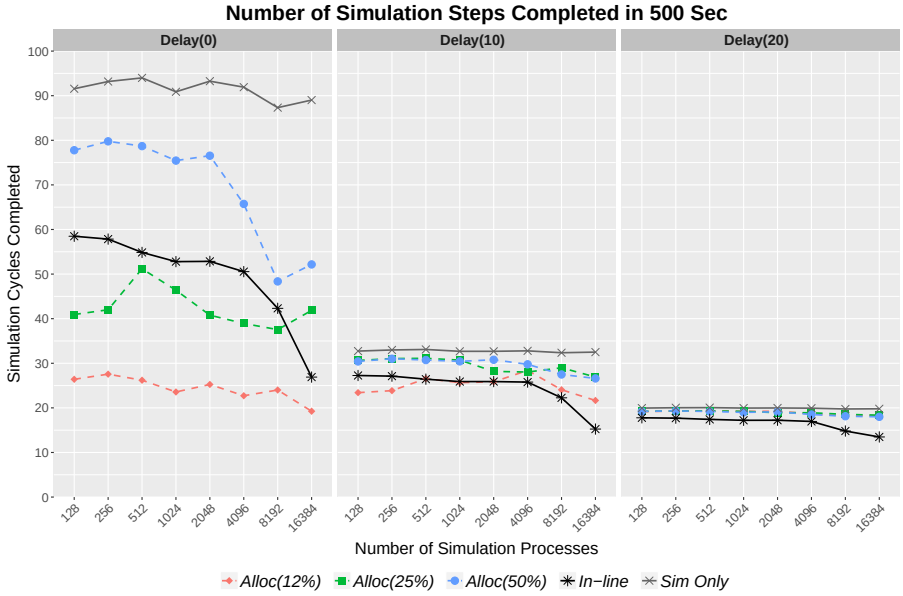


Fig. 5: Illustrating the cost of doing visualization. This figure plots the number of simulation cycles that could be completed in 500 seconds. The number of completed cycles are shown for the three simulation configurations using in-line visualization, and three configurations of in-transit visualization. This chart demonstrates that based on simulation time and resources, the simulation can proceed further with in-transit visualization vs. in-line visualization.

For example, a $Delay(0)$ configuration with 16384 simulation processors can complete 26 cycles using in-line visualization and 42 cycles using in-transit ($Alloc(25\%)$). This means that a 25% increase in compute power led to a 61% increase in productivity ($42/26 \times 100\% - 100\%$). Similarly, $Delay(0)$ and $Alloc(50\%)$ yields a 100% increase in productivity (26 cycles to 52 cycles) for 50% more resources, $Delay(10)$ and $Alloc(12\%)$ yields a 46% increase (15 cycles to 22 cycles) for 12% more resources, and $Delay(10)$ and $Alloc(25\%)$ yields an 80% increase (15 cycles to 27 cycles) for 25% more resources.

Figure 6 shows the total times for the $Delay(0)$ configuration where visualization was performed on every cycle, and every third cycle. For in-line visualization, the reduction in total time and reduction of frequency are nearly identical at 1/3. For the in-transit $Alloc(12\%)$ case, the reduction in total time is much more dramatic. When the visualization frequency is every simulation cycle, the simulation is stalled because there are not enough resources to keep up with the simulation. However, with a reduction in visualization frequency, the reduced allocation can keep up with the simulation, and the total time drops dramatically.

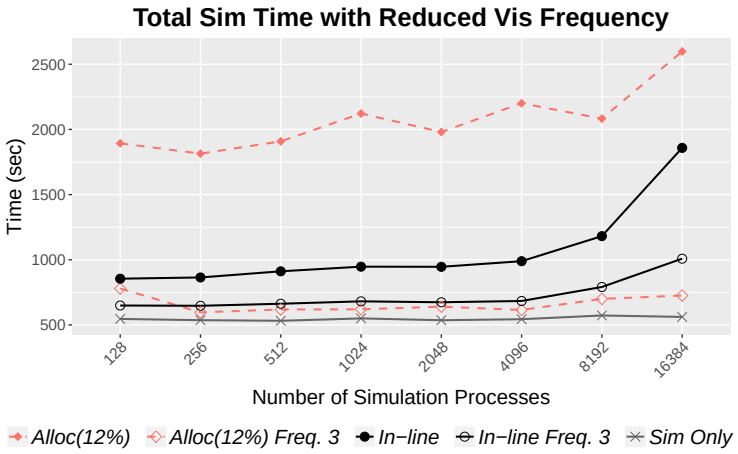


Fig. 6: Total execution time for the $Delay(0)$ simulation using in-line visualization and the $Alloc(12\%)$ in-transit visualization at two different frequencies, an image every step, and an image every third step. This chart demonstrates the large time savings that can be gained for in-transit visualization by reducing visualization frequency.

4.2 Total Cost

Figure 7 shows the cost of the node allocation for the selected configurations. We define this cost simply as $TotalTime \times TotalNodes$. This formulation takes into account that the in-transit method uses additional resources, allowing for the comparison to consider all resources used. In the in-transit case, because the simulation allocation is much larger than the visualization allocation, the costs are much higher where more simulation stalling occurs. This can be seen in the $Delay(0)$ configuration, particularly for $Alloc(12\%)$ in-transit visualization. In $Delay(10)$ and $Delay(20)$, we see nearly identical costs for the in-line and in-transit $Alloc(25\%)$ and $Alloc(50\%)$ configurations up to 8192 processes. For these cases, the extra resources pay for themselves.

The $Alloc(12\%)$ and $Delay(20)$ configuration is notable as the cost for in-transit becomes less than the cost for the in-line configuration at higher concurrency. This is a case where adding additional resources results in both a reduced time to solution, and a reduced allocation cost.

4.3 Scalability of Visualization Algorithms

The visualization pipeline used in this study consists of two operations: isocontouring and parallel rendering.

The isocontouring operation for both in-line and in-transit visualization cases selects values based on the minimum and maximum data values at each time step, which requires global communication of extents, i.e., two doubles per process. The cost of computing isocontours is a function of how much output geometry

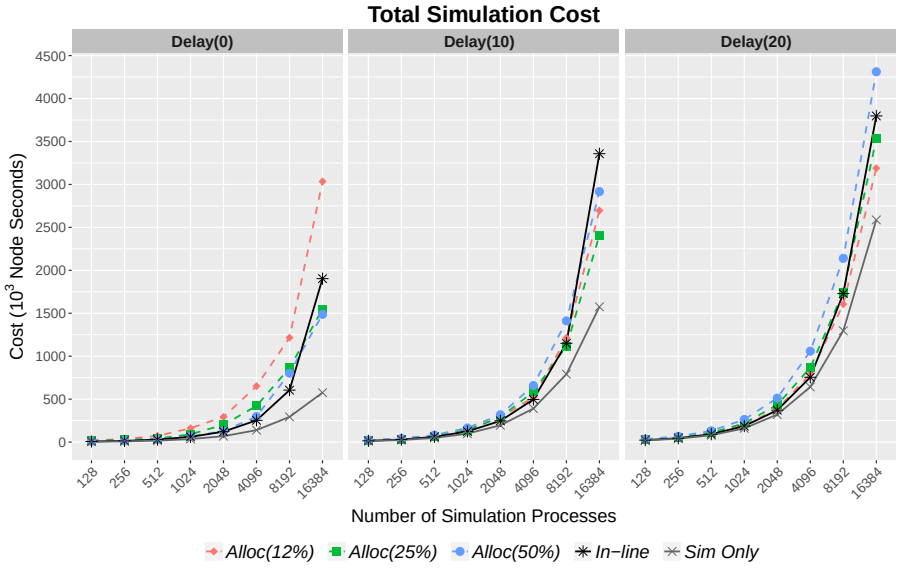


Fig. 7: Total cost in node-seconds to run the three simulation configurations using in-line visualization and three configurations of in-transit visualization.

is produced, which is dependent on the input data. Data blocks that do not contain the isovalues can be ignored. Workload imbalance is possible because the amount of work to perform is not the same for each data block.

The parallel rendering algorithm consists of two stages. First, each process renders the geometry produced by the isocontour operation, and second, these rendered images are combined using a parallel compositing algorithm to produce the final image. The parallel compositing algorithm requires significant communication.

Figure 8 shows the total time for rendering for simulation $Delay(0)$ using both in-line and in-transit paradigms. A sharp rise in rendering time occurs for in-line visualization at levels of concurrency above 2048. As described previously, the parallel rendering algorithm consists of two stages: rendering of data blocks, and parallel image compositing. The performance of in-line visualization is impacted by both stages of the parallel rendering algorithm. The input to the rendering are the isocontours generated in the previous step of the pipeline. The data blocks that contain more geometry will take more time to render. Likewise, the data blocks with less geometry will take less time to render.

For in-line visualization, each process has a single data block, and all of the processes will wait until the longest process is finished. Second, and more impactful, the performance of the parallel compositing algorithm is a function of the concurrency. Higher levels of concurrency require more communication, reducing the performance [24].

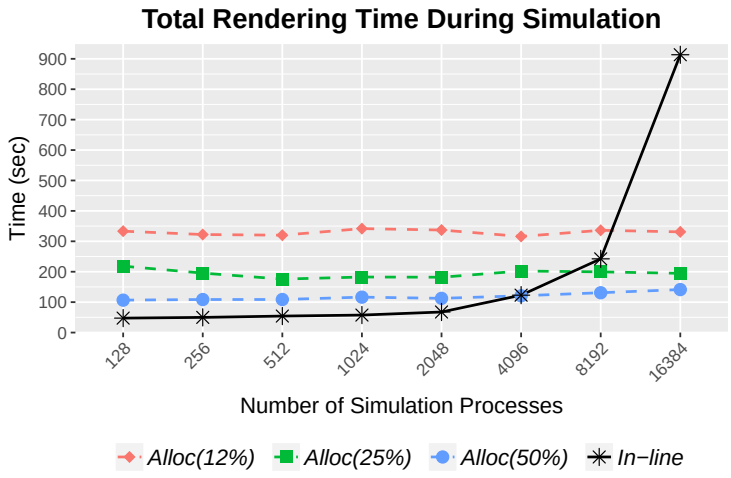


Fig. 8: Total time spent during rendering for the $Delay(0)$ configuration of the simulation for in-line and in-transit visualization. A communication bottleneck can be observed at high concurrency with in-line in situ as the time to perform rendering increases dramatically.

For in-transit visualization, a different situation exists. First, each visualization process is assigned multiple data blocks. When multiple blocks are assigned to each process, better load balancing is more likely to naturally occur. Second, and more impactful, the parallel compositing algorithm is run at lower concurrency, and so the performance is better.

Figure 9 shows a histogram of the idle times for the in-line and the $Alloc(50\%)$ in-transit case running on the $Delay(20)$ simulation at 16384 processors. The idle timings provide a higher-level look at the overall performance of the visualization operations. The idle time captures the amount of time each process spends waiting for other processes to complete. For in-line visualization, the histogram shape indicates significant idle time for a large number of processes. Note that there are a couple of in-line processes with little to no idle time, they are just not visible on this plot. On the other hand, the idle times for in-transit visualization lie much closer to zero, indicating much better load balancing across the entire visualization pipeline. These effects are the result of the load imbalance in iso-countouring and rendering, and the decreased scalability in parallel compositing that were described above.

5 Discussion

In this section we revisit the results presented in Section 4, and consider them in the broader context of the tradeoffs associated with in-line and in-transit in situ. Simulations, along with their requirements and resources, are unique. The same simulations could have different requirements based on the type of run

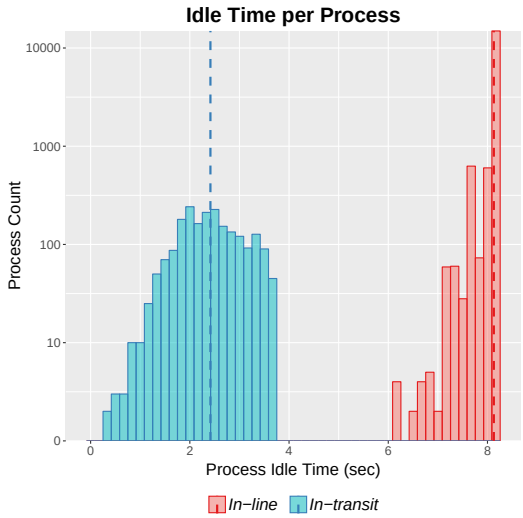


Fig. 9: Log scale histogram showing the idle time for each process during a single visualization step. The data shown are from the $Delay(20)$ simulation configuration run on 16384 processes, using in-line, and in-transit $Alloc(50\%)$ configurations for visualization. The dotted vertical lines give the mean value for both cases. This chart shows that in-line in situ causes higher per process idle times, driving up the total simulation time. Note that there are a couple of processes for each paradigm with no idle time, but they are not visible on this plot.

being performed, when the results are required, and the available resources. The major tradeoffs to consider are related to the time to solution (see Section 4.1) and the cost (see Section 4.2). When resources are available, time to solution might be the primary driver. Conversely, if resources are restricted, the cost might become the primary driver.

In Section 5.1 we discuss a cost model for both in situ visualization paradigms and provide some analysis that can help inform decisions. In Section 5.2 we discuss factors for consideration when time to solution is a primary driver.

5.1 In-line and In-transit Cost Models

The model for the cost of in-line visualization (C_V) can be described as:

$$C_V = (S + V)N_S \tag{1}$$

where S is the time to compute the simulation, V is the time to compute visualization, and N_S is the number of nodes used.

The model for the cost of in-transit visualization (C_T) can be described as:

$$C_T = (S + T_{IN})(N_S + N_D) \tag{2}$$

where S and N_S are as defined above. T_{IN} is the simulation stall time caused by transferring data to visualization resources, as well as any stall from the

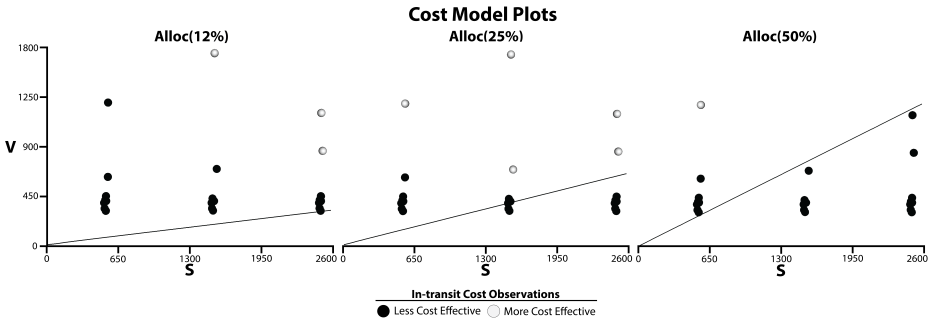


Fig. 10: Solutions of Equation 3 for values of T_{IN}. A contour line of $T_{IN} = 0$ is shown representing a data transfer and in-transit stall time of 0 seconds. Below this line in-transit visualization will never be viable from a cost perspective, given that any amount of transfer time will make it cost more than in-line. The data points for each experiment in the study are also shown, and indicate if the experiment cost more (black) or less (white) than the comparable in-line test.

simulation when waiting for the visualization to complete. N_D is the number of nodes used for visualization.

The costs of in-line and in-transit visualization are equal when equations 1 and 2 are equal. Setting them equal, and solving for T_{IN} gives:

$$T_{IN} = \frac{VN_S - SN_D}{(N_S + N_D)} \tag{3}$$

Given a simulation time (S), in-line visualization time (V), and a particular resource allocation (N_S and N_D), equation 3 gives the in-transit visualization data transfer time required for the costs of both paradigms to be equal. Smaller time values of T_{IN} will lower the cost of in-transit visualization with respect to in-line visualization. Conversely, larger time values of T_{IN} will raise the cost of in-transit visualization with respect to in-line visualization.

Figure 10 shows the solution to Equation 3 as a function of S and V for the fixed configurations (N_S and N_D) used in our study. The black line in each chart denotes where $T_{IN} = 0$ for each (S, V) pair for Equation 3. That is, in order for in-transit visualization to cost less than in-line visualization, the value of T_{IN}, must be zero. This is not physically possible, so (S, V) pairs below that line will always cost more using in-transit visualization. For (S, V) locations above the line, the in-transit visualization time must be less or equal to the value of T_{IN} in Equation 3 in order for the cost to be less than in-line visualization.

Data points from our study are also shown in Figure 10. For each configuration (N_S, N_D) in our study, the experiment generates values for S, V, and T_{IN}. Each point indicates an (S, V) data value, and the color of the point indicates if T_{IN} was less than (white) or greater than (black) the value in Equation 3. The slope of the $T_{IN} = 0$ contour provides an indicator of the performance requirements for in-transit visualization. As the slope increases (and resources used rises), increased performance is required.

5.2 In-line and In-transit Time to Solution

When time to solution is the primary driver, understanding the performance characteristics of the visualization and analysis algorithms used is important. If the algorithms scale well with respect to the number of simulation resources, in-transit visualization is likely to be slower, since it will be bottlenecked by both the network transfer time and the time to complete the analysis routine on the separate smaller set of resources. That is, if the visualization and analysis algorithms scale well, the time to perform them in-line will likely be faster than the time it would take to transfer the data across the network. Conversely, if the algorithm does not scale well, i.e. requires a lot of global or inter rank communication, then in-transit may be faster overall. This phenomenon of algorithms performing poorly at scale was demonstrated in our study as the parallel rendering was scaled up, as in Figure 8. In such situations, the reduced concurrency provided by the in-transit paradigm translates into significant time savings.

6 Conclusion and Future Directions

In this paper, we have presented a study that compares the performance of the two major in situ paradigms: in-line and in-transit visualization. We believe understanding tradeoffs in execution time and cost between these two paradigms are critical for the efficient use of in situ methods to handle the growing data problem. Without this understanding, it is difficult to make informed decisions when designing analysis and visualization workflows. If one technique significantly outperforms the other (in either time or cost), the community is likely to favor that technique. Further, if the techniques have similar performance, then other axes of consideration can be used in the decision making process.

This work provides two major contributions towards that end. First, we present a study and analysis for both paradigms on a simulation running on an HPC system at scale. We varied control parameters that define how each paradigm is configured, and analyze the performance tradeoffs for each. Second, we have explored various models for quantifying the costs of performing visualization using each in situ paradigm.

Further, our experiments gave insight into our hypothesis presented at the beginning of Section 3. First, we demonstrated that there are particular configurations for the simulation and analysis such that one in situ paradigm outperforms the other. This is a somewhat surprising result for the in-transit paradigm, as it means that allocating additional resources for analysis can lead to not just faster execution time for the simulation, but faster to the extent that there are fewer cycles used even when considering the additional resources. Second, we demonstrated that a communication heavy algorithm (parallel rendering) can cause bottlenecks when using an in-line paradigm at high concurrency, but by using a lower concurrency in-transit paradigm those bottlenecks would decrease. We further demonstrated that an in-transit paradigm can provide better load balancing for visualization algorithms. Lastly, we provided models that quantify

the cost of in situ visualization, and identified important relationships between the factors in each model and how they affect overall in situ cost.

In the future, we will perform follow up studies to better understand the behavior of both paradigms under different situations. These studies will include more visualization pipelines, different simulation codes, consider optimal numbers of visualization tasks to place per node in-transit, GPU's, and variations of both in-line and in-transit visualization that go beyond the common model. With this work we focused on a comparison based purely on time to solution and resource cost. There are additional factors of consideration [17] we would like to investigate in future work as well, as they provide insight into broader aspects when evaluating in situ visualization paradigms.

7 Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. This work was partially performed by UT-Battelle, LLC, with the US Department of Energy. This work was partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC (LLNL-CONF-769101).

References

1. Cloverleaf3d. http://uk-mac.github.io/CloverLeaf3D/, accessed: 2018-12-19
2. Adhinarayanan, V., Feng, W.c., Rogers, D., Ahrens, J., Pakin, S.: Characterizing and modeling power and energy for extreme-scale in-situ visualization. In: IEEE Parallel and Distributed Processing Symposium (IPDPS). pp. 978–987 (2017)
3. Ahern, S., et al.: Scientific Discovery at the Exascale: Report for the DOE ASCR Workshop on Exascale Data Management, Analysis, and Visualization (July 2011)
4. Ahrens, J., Geveci, B., Law, C.: ParaView: An End-User Tool for Large-Data Visualization. In: The Visualization Handbook, pp. 717 – 731 (2005)
5. Ayachit, U., Whitlock, B., Wolf, M., Loring, B., Geveci, B., Lonie, D., Bethel, E.W.: The SENSEI Generic In Situ Interface. In: Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV). pp. 40–44 (Nov 2016)
6. Ayachit, U., et al.: ParaView Catalyst: Enabling In Situ Data Analysis and Visualization. In: Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV). pp. 25–29 (2015)
7. Ayachit, U., et al.: Performance Analysis, Design Considerations, and Applications of Extreme-scale *In Situ* Infrastructures. In: ACM/IEEE Conference for High Performance Computing, Networking, Storage and Analysis (SC16) (Nov 2016)
8. Bauer, A.C., et al.: *In Situ* Methods, Infrastructures, and Applications on High Performance Computing Platforms, a State-of-the-art (STAR) Report. Computer Graphics Forum, Proceedings of Eurovis 2016 **35**(3) (Jun 2016)

9. Bennett, J.C., et al.: Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. pp. 49:1–49:9 (2012)
10. Childs, H., et al.: A contract-based system for large data visualization. In: Proceedings of IEEE Visualization 2005. pp. 190–198 (2005)
11. Childs, H., et al.: Extreme Scaling of Production Visualization Software on Diverse Architectures. IEEE Computer Graphics and Applications (CG&A) **30**(3), 22–31 (May/Jun 2010)
12. Dayal, J., et al.: Flexpath: Type-based publish/subscribe system for large-scale science analytics. In: IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing(CCGRID) (2014)
13. Docan, C., et al.: Dataspaces: an interaction and coordination framework for coupled simulation workflows. Cluster Computing **15**(2), 163–181 (2012)
14. Dorier, M., et al.: Damaris/viz: A nonintrusive, adaptable and user-friendly in situ visualization framework. In: IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV). pp. 67–75 (Oct 2013)
15. Gamell, M., et al.: Exploring power behaviors and trade-offs of in-situ data analytics. In: International Conference on High Performance Computing, Networking, Storage and Analysis (SC). pp. 1–12 (2013)
16. Goodale, T., et al.: The cactus framework and toolkit: Design and applications. In: VECPAR 2002. pp. 197–227
17. Kress, J., et al.: Loosely coupled in situ visualization: A perspective on why it's here to stay. In: Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV). pp. 1–6 (Nov 2015)
18. Larsen, M., et al.: The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In: Workshop on In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV). pp. 42–46 (2017)
19. Liu, Q., et al.: Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. Concurrency and Computation: Practice and Experience **26**(7), 1453–1473 (2014)
20. Mallinson, A., et al.: Cloverleaf: Preparing hydrodynamics codes for exascale. The Cray User Group **2013** (2013)
21. Moreland, K., et al.: VTK-m: Accelerating the Visualization ToolKit for Massively Threaded Architectures. Computer Graphics & Applications **36**(3), 48–58 (2016)
22. Oldfield, R.A., Moreland, K., Fabian, N., Rogers, D.: Evaluation of methods to integrate analysis into a large-scale shock shock physics code. In: Proceedings of the 28th ACM international conference on Supercomputing. pp. 83–92. ACM (2014)
23. Parker, S., Johnson, C.: SCIRun: a scientific programming environment for computational steering. In: ACM/IEEE Conference on Supercomputing. p. 52 (1995)
24. Peterka, T., Ma, K.L.: Parallel image compositing methods. In: High Performance Visualization: Enabling Extreme-Scale Scientific Insight (2012)
25. Rodero, I., et al.: Evaluation of in-situ analysis strategies at scale for power efficiency and scalability. In: IEEE/ACM Symposium on Cluster, Cloud and Grid Computing (CCGrid). pp. 156–164 (2016)
26. Whitlock, B., Favre, J., Meredith, J.: Parallel in situ coupling of simulation with a fully featured visualization system. In: In Proceedings of the 11th Eurographics conference on Parallel Graphics and Visualization. pp. 101–109 (2011)
27. Zhang, F., Jin, T., Sun, Q., Romanus, M., Bui, H., Klasky, S., Parashar, M.: In-memory staging and data-centric task placement for coupled scientific simulation workflows. Concurrency and Computation: Practice and Experience **29**(12) (2017)