Opportunities for Cost Savings with In-Transit Visualization

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. We analyze the opportunities for in-transit visualization to provide cost savings compared to in-line visualization. We begin by developing a cost model that includes factors related to both in-line and in-transit which allows comparisons to be made between the two methods. We then run a series of studies to create a corpus of data for our model. We run two different visualization algorithms, one that is computation heavy and one that is communication heavy with concurrencies up to 32,768 cores. Our primary results are in exploring the cost model within the context of our corpus. Our findings show that in-transit consistently achieves significant cost efficiencies by running visualization algorithms at lower concurrency, and that in many cases these efficiencies are enough to offset other costs (transfer, blocking, and additional nodes) to be cost effective overall. Finally, this work informs future studies, which can focus on choosing ideal configurations for in-transit processing that can consistently achieve cost efficiencies.

1 Introduction

In situ visualization is increasingly necessary to address I/O limitations on supercomputers [2,3]. That said, the processing paradigm for in situ visualization can take multiple forms. With this study, we consider two popular forms. In the first form, which we refer to in this paper as in-line visualization, the visualization routines are embedded into the simulation code, typically via a library which is linked into the simulation binary. In this case, the visualization routines directly access the memory of the simulation code. When it is time to perform visualization tasks, the simulation pauses, and the visualization tasks use the same nodes that were being used for the simulation. With the second form, which we refer to in this paper as in-transit visualization, extra compute nodes run concurrently to the simulation. In this case, the simulation runs on the primary compute nodes (the "simulation nodes") and the visualization runs, as a separate program, on the extra compute nodes (the "in-transit nodes"). The simulation

P. Sadayappan et al. (Eds.): ISC High Performance 2020, LNCS 12151, pp. 146–165, 2020.
https://doi.org/10.1007/978-3-030-50743-5_8

shares data with the visualization program by sending data over the network. In this scenario, the simulation and visualization are both running at the same time.

In-line and in-transit both have beneficial aspects [6]. For example, in-transit naturally lends itself to fault tolerance, while in-line saves on usage of primary memory. In short, there are good reasons motivating the use of either technique. However, one factor has a special importance, namely cost. With this study, we define cost to be in units of "node seconds," i.e., using ten compute nodes for one second or one compute node for ten seconds are both "ten node seconds." Cost directly informs the size of the request needed on a supercomputer to perform the simulation. Therefore, we believe understanding the relative costs of in-transit and in-line is critical in helping scientists determine which paradigm to use. While other factors (fault tolerance, memory usage, etc.) may play a role in the decision, we believe cost will be a critical factor.

In-transit visualization incurs new costs that do not exist with in-line visualization. There are additional resources for the in-transit nodes, and a new activity to perform: transferring the data from the simulation nodes to the in-transit nodes. Further, if the in-transit nodes are not able to perform their tasks quickly enough, they can block the simulation from advancing. While blocking the simulation is not the only possible decision for this scenario, it is the decision we consider in the context of this paper.

Despite these additional costs, in-transit also has a potential cost advantage that in-line does not have. The number of in-transit nodes is typically much less than the number of simulation nodes. Further, when algorithms exhibit poor scaling, fewer nodes are more efficient. In effect, in-transit has the potential to reduce costs that result from poor scaling of visualization algorithms. Consider a scenario: if a visualization algorithm takes 1 s on 1000 nodes running in-line, but the same algorithm takes 50 s on 10 nodes running in-transit, then the visualization cost is 1000 node seconds for in-line and 500 node seconds for in-transit. We define a term to capture this phenomenon: **Visualization Cost Efficiency Factor (VCEF)**. $VCEF$ is the in-line visualization cost divided by the in-transit visualization cost. In the scenario just described, the $VCEF$ would be 1000/500 or 2—the cost to perform in-line is 2X more than in-transit. Of course, $VCEF$ is just one consideration for in-transit, and must be considered alongside its other factors, including, extra resources, transfer costs, and blocking, which impose barriers to cost savings.

Our hypothesis entering this study is that there are configurations of in-transit visualization such that the cost to reach the final solution are less in-transit than in-line. To that end, for this study, we consider the topic of relative costs between in-transit and in-line visualization. What makes our study novel is the identification and usage of $VCEF$. We observe that $VCEF$ is a significant phenomenon; our communication-heavy algorithm regularly yields a $VCEF$ of four or above, and even our computation-heavy algorithm yields such values at very high concurrency. This high $VCEF$ value in turn allows in-transit to become cost effective overall in many scenarios, as the savings are enough to offset other

costs (extra resources, transfer costs, and blocking). We also provide a model for reasoning about this space, and a corpus of data that reflects experiment times for currently popular software. Overall, this study provides significant evidence that in-transit can be cost effective.

2 Related Works

In recent years, some application teams have began seeing the need to adopt the in situ approach for visualization and analysis of large-scale simulations [10]. One strength of in situ methods is the ability to access all of the data during the course of a simulation, and only save what is interesting. This means, in situ is not just a tool for visualization, but also for processing of data such as reduction, explorable feature extractions, simulation monitoring, and the generation of statistics [11]. The choice then, comes down to which in situ approach is appropriate for a given application.

To aid in making that choice, several studies have looked at in-transit and in-line from the perspective of time. Morozov et al. [16] describes a system for launching in situ/in-transit analysis routines, and compares each in situ technique based on time to solution for two different analysis operations. They find there were times when in-transit analysis was faster due to how the analysis code scaled. Friesen et al. [5] describes a setup where in-line and in-transit visualization are used in conjunction with a cosmological code to run two different analysis routines. They analyze the time to solution using both in situ techniques, finding that there are configurations where in-transit is faster to use, due to the inter-node communication overhead of the analysis routines. These and other studies have largely focused on analysis pipelines which can have different communication and computation scaling curves than visualization pipelines. Further, they do not do an in depth analysis of the trade-offs associated with in-transit or in-line methods.

Our work takes a different view than these past works. First, we concentrate on in situ visualization pipelines. Second, we focus specifically on in-line in situ vs. in-transit in situ from the perspective of visualization frequency, resource requirements, and how different combinations of these factors impact the final cost of the simulation and visualization for research scientists.

There are three highly relevant works preceding this work:

- Oldfield et al. [17] also considered in-transit and in-line costs. The main difference between their work and our own is that they focused on analysis tasks which did not benefit from a $VCEF$ speedup. As such, their findings differ from ours.
- Malakar et al. did twin studies on cost models, one for in-line [12] and one for in-transit [13]. Once again, these studies did not consider $VCEF$. Further, they considered optimizing allocation sizes and analysis frequencies which is a complementary task to our effort.

– Work by Kress et al. [7] considered trade-offs between in-transit and in-line for isosurfacing at high concurrencies. This study was the first to show evidence of $VCEF$. However, the algorithm considered was computation-heavy, so the extent of the effect was smaller and only appeared at very high concurrency. Further, that paper lacked a cost modeling component, rather just observing that the phenomenon was possible. Our paper focuses exclusively on cost savings, providing a model and considering both computation- and communication-heavy visualization algorithms. Finally, we note the corpus of data for our study in part draws on runs from the Kress et al. study.

3 Cost Model

This section defines a cost model for determining when in-transit visualization can cost less than in-line visualization. First, terms are introduced for the operations that occur in both in-line and in-transit visualization. Next, we use those terms to demonstrate when in-transit will cost less than in-line visualization, and provide a discussion for when and how this occurs. Finally, we derive a formulation to determine the degree of scalability of in-transit over in-line, ($VCEF$), that is required for in-transit to be cost effective.

3.1 Definition of Terms

Below we define terms for both in-line and in-transit visualization operations.

– Let T be the time for the simulation to advance one cycle.
– Let N be the number of nodes used by the simulation code.
– Let Res_p be the proportion of nodes (resources) used for in-transit visualization. E.g., if the number of nodes for the simulation (N) is 10,000 and the number of nodes for in-transit visualization is 1,000, then $Res_p = 1,000/10,000$, which is 0.1.
– Let Vis_p be the proportion of time spent doing visualization in the in-line visualization case. E.g., if T is 5 s and the in-line visualization time is 1 s, then $Vis_p = 1/5$, which is 0.2.
– Let $Block_p$ be the proportion of time that the simulation code is blocking while waiting for in-transit visualization to complete. E.g., if T is 5 s and the simulation has to wait an additional 2 s for the in-transit resources to complete, then $Block_p = 2/5$, which is 0.4. If the in-transit visualization completes and does not block the simulation, then $Block_p$ is 0.
– Let $VCEF$ be the term identified earlier in this paper that captures the efficiency achieved by running at lower concurrency. E.g., if in-line visualization took 1 s on 10,000 nodes, but in-transit visualization took 5 s on 1,000 nodes, then $VCEF$ would be $\frac{1\times10,000}{5\times1,000}$, which is 2.

We have two terms for transferring data because sending data from the simulation side may be faster than receiving it on the in-transit side. For example, if 8 simulation nodes send to 1 visualization node, then that 1 visualization node will need to unserialize eight times as much data as each of the simulation nodes serialized.

- Let $Send_p$ be the proportion of time by the simulation code sending data to in-transit visualization resources. E.g., if T is 5 s and the send time is 2 s, then $Send_p = 2/5$, which is 0.4.
- Let $Recv_p$ be the proportion of time spent receiving data on the in-transit visualization resources. E.g., if T is 5 s and the transfer time is 2 s, then $Recv_p = 2/5$, which is 0.4.

3.2 Base Model Defined

We define our base cost model below. This cost model will be refined in Sect. 3.4 as we consider the implications of blocking. The cost for in-transit visualization will be lower than in-line visualization when:

$$\begin{gathered}
\text{(total resources with in-transit)} \times \text{(time per cycle for simulation with in-transit)} \\
< \\
\text{(total resources with in-line)} \times \text{(time per cycle for simulation with in-line)} \\
\Longrightarrow \\
\text{(\#in-transit nodes + \#simulation nodes)} \times \\
\text{(simulation cycle time + transfer time + block time)} \\
< \\
\text{(\#simulation nodes)} \times \text{(simulation cycle time + in-line vis time)}
\end{gathered} \tag{1}$$

Using the terms defined above in Sect. 3.1, this becomes:

$$(N \times Res_p + N) \times (T + T \times Send_p + T \times Block_p) < (N) \times (T + T \times Vis_P) \tag{2}$$

This equation can be simplified by dividing both sides by the simulation cycle time (T) and number of nodes (N):

$$(1 + Res_p) \times (1 + Send_p + Block_p) < (1 + Vis_P) \tag{3}$$

If Eq. 3 is true, then in-transit costs less than in-line.

3.3 Base Model Discussion

In-transit visualization has three different costs that do not occur with in-line. (1) In-transit visualization requires data transfer, which slows down the simulation nodes. (2) In-transit visualization requires dedicated resources beyond those required for in-line. If the in-transit visualization finishes quickly, these additional resources sit idle, and yet still incur cost. (3) In-transit can block the simulation if the visualization is not finished before the simulation is ready to send data for the next cycle. This is very harmful since it slows down the simulation nodes. There are alternatives to blocking, for example skipping cycles, and only visualizing the latest. In this study, our focus is on blocking, and we do not consider the alternatives.

Given the three additional costs incurred by in-transit, the *only* way for it to cost less than in-line is for the visualization to run faster at lower concurrency. In other words, the cost savings with in-transit can *only* occur if the benefit of

($VCEF$) outweighs the combined effects of the three additional costs described above. The fact that certain operations are more efficient at lower levels of concurrency provides an opporunity for a more cost effective solution.

That said, there are scenarios where any value of $VCEF$ is insufficient to achieve cost savings. Examples where in-transit can never be more cost effective, regardless of $VCEF$, are discussed below:

- If blocking takes longer than in-line visualization (e.g., $Block_p = 0.3, Vis_p = 0.2$), it is impossible to be more cost efficient. For example, even if $T = \epsilon$, then $(1 + \epsilon) \times (1 + \epsilon + 1.3) < (1 + 1.2)$ is not possible.
- Further, even if $Block_p = 0$ (no blocking), then some in-transit configurations will still always be less efficient:
  - if the simulation transfer cost is bigger than the in-line visualization time (e.g., $Send_p = 0.4, Vis_p = 0.2$), then: $(1 + \epsilon) \times (1 + 0.4 + 0) < 1.2$
  - if there are many in-transit nodes (e.g., $Res_p = 0.5$) and the in-line visualization time is sufficiently fast (e.g., $Vis_p = 0.5$), then: $(1 + 0.5) \times (1 + \epsilon + 0) < 1 + 0.5$

3.4 When Does Blocking Occur?: Replacing $Block_p$ via $VCEF$

In this section we expand the model by using the $VCEF$ term to determine when blocking will occur. We then present two new equations that define when in-transit will cost less if blocking does or does not occur.

Consider what it means to block. Blocking occurs when in-transit resources are taking longer to do their job than the simulation resources are taking to do their job. Similarly, "not blocking" means that the in-transit resources are doing their job faster than the simulation resources take to do their job. So, what does "time to do their job" mean? For the simulation side, this means the time to advance the simulation plus the time to send the data, i.e., $T + T \times Send_p$. For the in-transit side, this means the time to receive data ($T \times Recv_p$) plus the time to do the visualization task. This latter time is explored below.

Nominally, assuming that visualization scaled perfectly as a function of concurrency, the cost (number of node seconds) to do the visualization task can be directly calculated from the in-line case: $N \times (Vis_p \times T)$. However, a key premise of this study is that in-transit has an advantage at lower concurrency because of $VCEF$. Because in-transit is running at a lower concurrency, the cost is scaled by the $VCEF$ term: $\frac{N \times (Vis_p \times T)}{VCEF}$. Finally, the time to carry out the visualization task on the in-transit nodes would be the $VCEF$-reduced cost divided by the resources ($N \times Res_p$). Thus, the in-transit visualization time is:

$$\frac{N \times (T \times Vis_p)}{VCEF \times N \times Res_p} \tag{4}$$

Canceling out N gives a simpler form:

$$\frac{Vis_p \times T}{VCEF \times Res_p} \tag{5}$$

Restating, blocking occurs with in-transit when the time to receive data plus the visualization time is greater than the simulation time plus the time to send data:

$$Recv_p \times T + \frac{Vis_p \times T}{VCEF \times Res_p} > T \times (1 + Send_p) \tag{6}$$

This means that blocking *does not* occur if:

$$Recv_p \times T + \frac{Vis_p \times T}{VCEF \times Res_p} \leq T \times (1 + Send_p) \tag{7}$$

The terms in Eq. 7 can be rearranged to find the $VCEF$ values when blocking *does not* occur:

$$\frac{Vis_p}{Res_p \times (1 + Send_p - Recv_P)} \leq VCEF \tag{8}$$

This analysis on blocking informs the original question: when does in-transit incur less cost than in-line? This can be answered using a combination of Eq. 3 and our observations about blocking in this section. If blocking does not occur, then $Block_p$ drops out as zero, and Eq. 3 is simplified:

$$(1 + Res_p) \times (1 + Send_p) < (1 + Vis_P) \tag{9}$$

If blocking does occur, then the simulation advances only as fast as the in-transit resources can take new data. This means that the time term for the left-hand side of Eq. 3, which was previously $1 + Send_p$, is replaced with the in-transit time. Using the relationship in Eq. 6, we get:

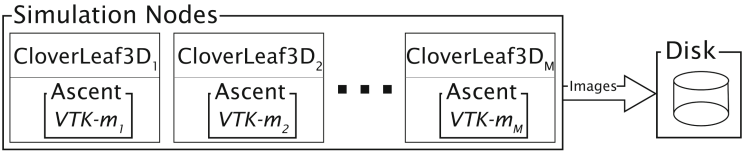
$$(1 + Res_p) \times (Recv_p + \frac{Vis_p}{VCEF \times Res_p}) < (1 + Vis_P) \tag{10}$$

3.5 Cost Model Discussion

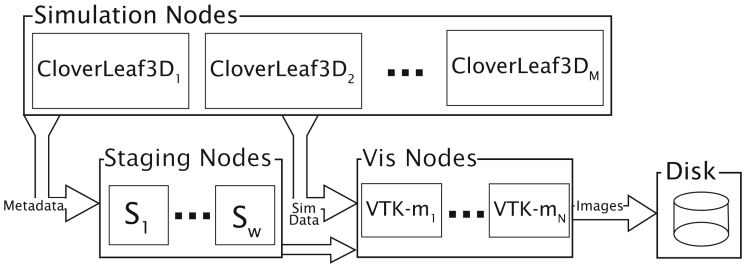
The basis of the cost model are described above in Eqs. 3, 8, 9, and 10. This model allows the relative costs of in-line and in-transit visualization for a particular configuration to be analyzed. The first step is to determine the cost feasibility of in-transit. Equation 3 serves as a threshold for determining when this is *possible*. If Eq. 3 is false, in-line visualization is the cost-effective solution. Otherwise, when Eq. 3 is true, Eqs. 8, 9, and 10 are used to determine cost feasibility based on blocking, as follows:

– The $VCEF$ value necessary to prevent blocking is given by Eq. 8:
$VCEF \geq \frac{Vis_p}{Res_p \times (1 + Send_p - Recv_P)}$

- For cases when there is no blocking, using Eq. 9 shows that in-transit is cost efficient if:
$(1 + Res_p) \times (1 + Send_p) < (1 + Vis_P)$
- Otherwise, for cases where blocking occurs, using Eq. 10 shows that in-transit is cost efficient if:
$(1 + Res_p) \times (Recv_p + \frac{Vis_p}{VCEF \times Res_p}) < (1 + Vis_P)$



(a) In-line visualization setup. The simulation and visualization alternate in execution, sharing the same resources.



(b) In-transit visualization setup. 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.

4 Corpus of Data

In this section we detail the experimental setup, methods, and software used to generate our corpus of data, as well as a cursory overview of the data we collected.

4.1 Experiment Software Used

To generate data for this study, we use CloverLeaf3D [1,14], a hydrodynamics proxy-application. Cloverleaf3D spatially decomposes the data uniformly across distributed memory processes, where each process computes a spatial subset of the problem domain. To couple CloverLeaf3D with both in-transit and in-line in situ, we leveraged existing integrations with Ascent [8].

In-line visualization is accomplished with Ascent which uses VTK-m [15] for visualization operations. The visualization is described through a set of actions which Ascent turns into a data flow graph, and then executed. Figure 1a depicts how the software components interact in the in-line workflow.

In-transit visualization used Ascent's integration with the Adaptable I/O System (ADIOS) [9] to transport data from the simulation nodes to the in-transit nodes using its RDMA capabilities [4,18]. ADIOS requires the use of dedicated staging nodes to hold the metadata necessary to service RDMA requests. Once the data are transported, the visualization tasks are performed using VTK-m. To be clear, the same VTK-m code was being used for both in-line and in-transit visualization. The only differences are the number of nodes used for visualization,

the shared-memory parallelism approach (see Sect. 4.5), and the use of ADIOS for data transport to a separate allocation.

Figure 1b depicts how the software components interact in the in-transit workflow.

4.2 Visualization Tasks Studied

There were two classes of visualization tasks in this study, computation heavy and one that is communication heavy. The computation heavy task was isocontouring and parallel rendering, while the communication heavy task was volume rendering. Visualization was performed after each simulation step. The computation heavy task consisted of creating two isocontours at values of 33% and 67% between the minimum and maximum value of the simulations *energy* variable, followed by ray trace rendering. The ray tracing algorithm first locally rendered the data it contained, then all of the locally rendered images were composited using radix-k. The communication heavy task consisted of volume rendering the simulations *energy* variable. Compositing for volume rendering is implemented as a direct send.

4.3 Application/Visualization Configurations

In this study we used five different in situ configurations of the application and visualization:

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

For in-transit visualization, predetermined percentages of simulation resources for visualization were selected. These percentages, were selected based off of a rule of thumb where simulations typically allow up to 10% of resources for visualization. 10% was our starting point, and we then selected two additional higher allocations to explore a range of options. We 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. We leave a lower percentage study as future work on a future machine. Finally, we ran each one of these configurations with weak scaling with concurrency ranging between 128 and 32,768 processes, with 128^3 cells per process (268M cells to 68B cells).

CloverLeaf3d uses a simplified physics model, as such, it 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. We simulated longer cycle times by configuring 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. We used three different levels of delay:

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

Lastly, we ran each test for 100 time steps using a fixed visualization frequency of once every time step. This frequency ensures that fast evolving structures in simulation data are not missed. Also, very frequent visualization gives us an upper bound for how visualization will impact the simulation.

4.4 Hardware

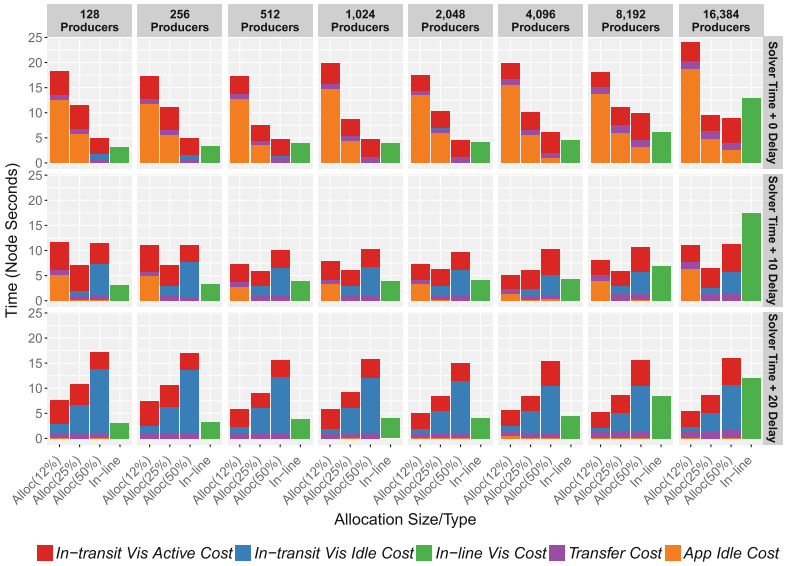
All runs in this study were performed on the Titan supercomputer deployed at the Oak Ridge Leadership Computing Facility (OLCF). Because the mini-app we used for our study runs on CPUs only, we restricted this study to simulations and visualizations run entirely on the CPU.

4.5 Launch Configurations

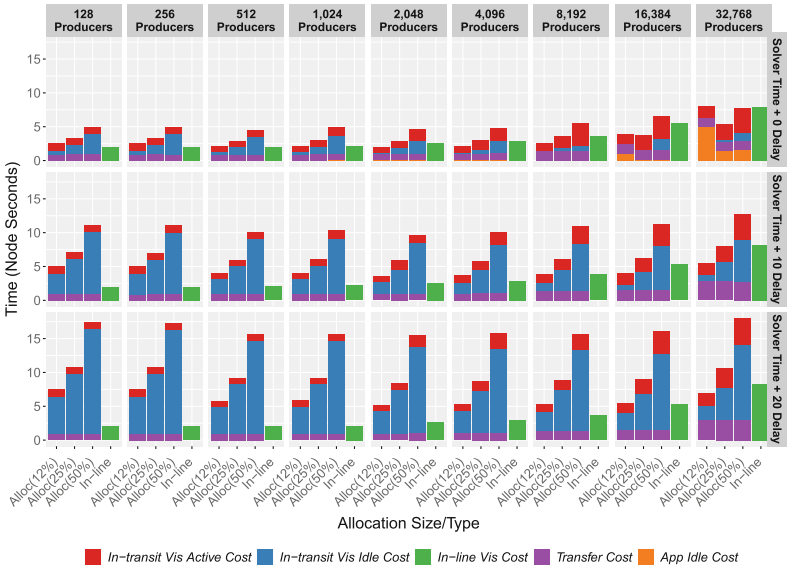
The configuration for each experiment performed is shown in Table 1. Isosurfacing plus rendering was run on up to 16K cores, volume rendering was run on up to 32K cores. Because CloverLeaf3D is not an OpenMP code, the in-line in situ and the simulation only configurations were launched with 16 ranks 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. Additionally, the in-transit configuration required the use of dedicated staging nodes to gather the metadata from the simulation in order to perform RDMA memory transfers from the simulation resource to the visualization resource. These additional resources are accounted for in Table 1 and are used in the calculation of all in-transit results.

Table 1. Resource configuration for each experiment in our scaling study.

Test	**Sim Procs**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**
Configuration	**Data Cells**	648^3	816^3	1024^3	1296^3	1632^3	2048^3	2592^3	3264^3	4096^3
In-line	**Total Nodes**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
In-transit *Alloc*(12%)	Vis Nodes	1	2	4	8	16	32	54	128	256
	Staging Nodes	1	2	2	4	4	8	8	16	16
	Total Nodes	**10**	**20**	**38**	**76**	**148**	**296**	**584**	**1168**	**2320**
In-transit *Alloc*(25%)	Vis Nodes	2	4	8	16	32	64	128	256	512
	Staging Nodes	1	2	2	4	4	8	8	16	16
	Total Nodes	**11**	**22**	**42**	**84**	**164**	**328**	**648**	**1296**	**2576**
In-transit *Alloc*(50%)	Vis Nodes	4	8	16	32	64	128	256	512	1024
	Staging Nodes	1	2	2	4	4	8	8	16	16
	Total Nodes	**13**	**26**	**50**	**100**	**196**	**392**	**776**	**1552**	**3088**



(a) Cost breakdown for the isosurfacing and rendering tests.



(b) Cost breakdown for the volume rendering tests.

Fig. 2. Stacked bar charts comparing the total cost per step for using in-transit and in-line visualization. In-transit visualization is broken down into cost for the time that the visualization is actively working, cost for the time that it is idle, cost for the time it is receiving data from the simulation, and cost associated with blocking the application. The application active cost is excluded from this chart as it is the same for each level of Delay, and obfuscates the times for visualization and data transfer.

An important detail from these configurations is that the in-line tests have one core per MPI task, while the in-transit tests have four cores per MPI task. Where the in-line tests were restricted to the MPI approach of the simulation code, the in-transit tests were not, enabling shared-memory parallelism. As a result, in-transit had even fewer participants in communication, which will boost its $VCEF$ factor.

4.6 Overview of Data Collected

In total, we ran 255 individual tests, each for 100 time steps. From each of these tests we collected the total time for each time step from both the simulation and visualization resources, as well as more fine grained timers placed around major operations. After the runs were complete, the total cost was calculated by multiplying the total time by the total number of nodes listed in Table 1. Figure 2a shows the total cost per time step we observed for each of the isosurfacing plus rendering tests and Fig. 2b shows the total cost per time step we observed for each of the volume rendering tests. These charts break down the cost of each step associated with each of our runs, showing if the simulation was blocked by the visualization and how much that blocking cost, how much it cost to transfer data from the simulation to the in-transit resources, how long the visualization resources were active and their cost, how long they were idle and that cost, and how long the in-line visualization operation took and its associated cost.

There are marked differences in the performance of the isosurfacing and rendering runs versus the volume rendering runs. The isosurfacing tests have large periods of blocking whereas the volume rendering runs have very little. One reason for that blocking was that on average, isosurfacing and rendering took twice as long per step as volume rendering. Finally as the simulation cycle time increased, isosurfacing and rendering benefited more than volume rendering, showing that the isosurfacing tests were compute bound on the in-transit resources.

5 Results

In this section we use the model described in Sect. 3 to analyze the data collected from our experiments. In particular, we follow the discussion detailed in Sect. 3.5. In Sect. 5.1, we discuss and analyze the magnitude of $VCEF$ (Eq. 8) for each experiment. In Sect. 5.2 we use Eq. 3 from our model to determine the in-transit cost savings feasibility for each experiment. Finally, in Sect. 5.3, we combine these two and discuss the experiments that are feasible and have sufficient $VCEF$ to produce cost savings using in-transit for both non-blocking and blocking cases (Eqs. 9 and 10).

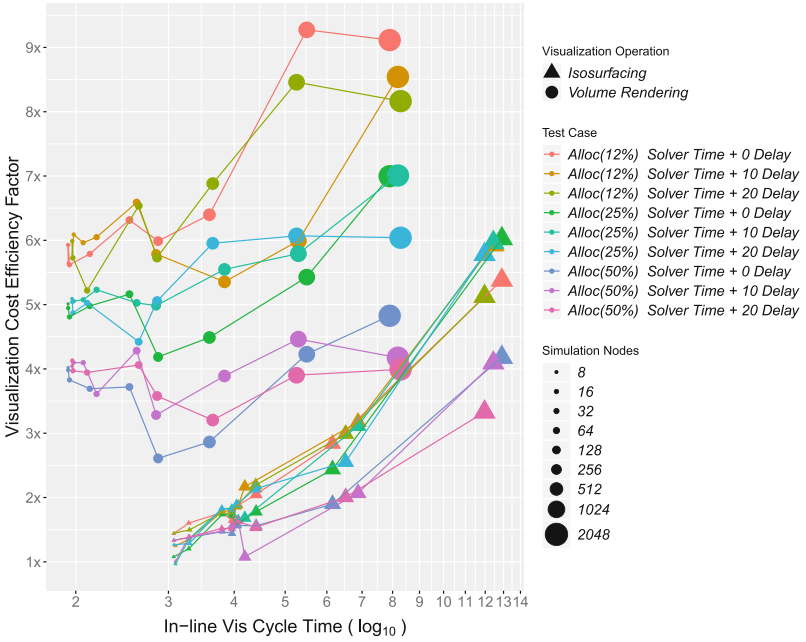


Fig. 3. This plot shows in-transit $VCEF$ as a function of the in-line cycle time. Isosurfacing experiments are denoted with a triangle glyph and volume rendering with a circle glyph. Each glyph is scaled by the concurrency of the experiment (isosurfacing 8-1024; volume rendering: 8-2048). Experiments are group by color (configuration) and connected by lines (concurrency sequence).

5.1 *VCEF* Magnitude Across Experiments

Figure 3 shows the $VCEF$ for each experiment. We felt the most surprising result was how large $VCEF$ values were as a whole. Many of the experiments had values above $4X$, which creates significant opportunities for the cost effectiveness of in-transit. Surprisingly, volume rendering experiments where the in-transit resources were 50% of the simulation ($Alloc(50\%)$) were able to achieve $VCEF$ values of about $4X$. Putting this number in perspective, if an $Alloc(50\%)$ experiment runs in the same amount of time as its in-line counterpart using half the concurrency, then its $VCEF$ would be 2. This is because it would have run using half the resources while taking the same amount of time as in-line. Higher values indicate that the runtime has decreased at smaller concurrency, i.e., 4X cost efficiency via using half the resources and running 2X faster. Further, we note this volume rendering algorithm has been extensively optimized and is used in a production setting. This result highlights the significant advantage that $VCEF$ provides. Algorithms with poor scalability (i.e., heavy communication) are able to run at lower levels of concurrency, and therefore achieve better performance.

As expected, $VCEF$ is heavily dependent on the type of algorithm. The volume rendering experiments were communication-heavy, lending itself to higher

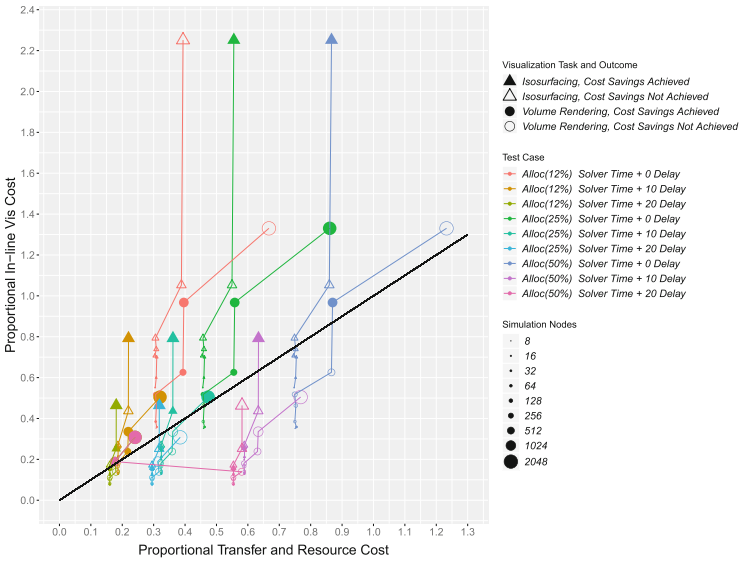
cost efficiency when running at lower concurrency. The isosurfacing experiments were computation-heavy—first, an isosurface is calculated, and then it was rendered. The isosurface calculation is embarrassingly parallel, so there is no reason to expect a high $VCEF$. That said, the parallel rendering became very slow at high concurrency, as evidenced by the high in-line times (>10 s). This was due to the communication required to perform the image compositing and the final reduction using the radix-k algorithm. In these cases, the $VCEF$ values increased from $3X$ to $6X$.

While the main takeaway of Fig. 3 is high $VCEF$ values, a secondary takeaway looks ahead to our analysis of cost savings, and in particular establishing intuition about which configurations will be viable for cost savings. All volume rendering experiments had high $VCEF$ values, while only isosurfacing experiments at very high concurrency had high $VCEF$ values. The isosurfacing experiments at lower concurrencies had smaller $VCEF$ values, which makes them less likely to offset the additional costs incurred for in-transit (transfer times, blocking, idle).

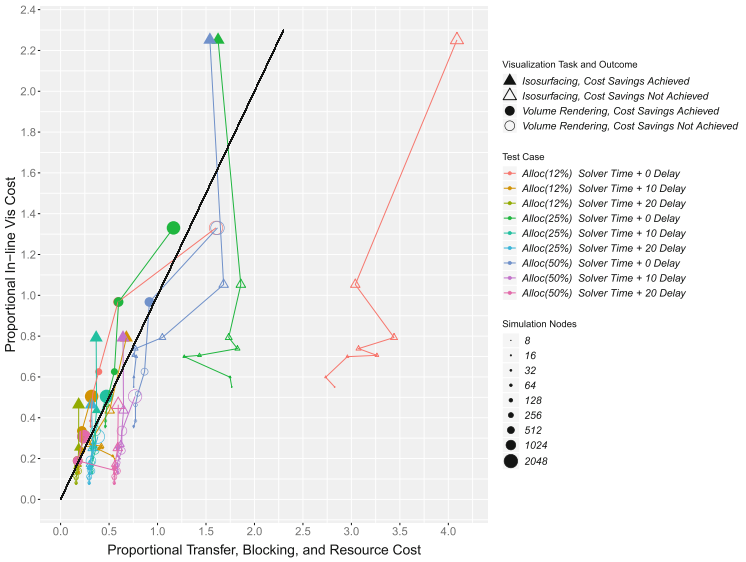
5.2 Feasibility of Cost Savings

Equation 3 from our model is used to determine the feasibility of cost savings for in-transit visualization. When Eq. 3 is true, then cost feasibility is possible. Figure 4a uses this equation to show the feasibility for each experiment. The black line shows where in-line and in-transit costs are identical, and the region above the black line is cost feasibility for in-transit. This figure follows discussion from Sect. 3.3. For example, if the in-line cost is less than the transfer cost, then no $VCEF$ value can make in-transit cost effective. Or if the resources devoted to in-transit are very large, then they will likely sit idle and be a incur cost at no gain. About half of our experiments were in this category, incapable of achieving cost savings with in-transit, because the transfer and resource costs exceeded the in-line costs. In the remaining half of the experiments, our choice for the number of in-transit nodes created a potentially feasible situation—the resources dedicated to in-transit and the cost of transferring data was less than the in-line visualization cost. That said, only some of these experiments actually led to cost savings with in-transit. This is because the feasibility test for Fig. 4a placed no consideration on whether the in-transit resources were sufficient to perform the visualization task. In some cases, $VCEF$ was enough that the in-transit resources could complete its visualization task within the allotted time. In others cases, $VCEF$ was not sufficient, and this caused the in-transit resources to block. Figure 4b takes this blocking into account, and faithfully plots the terms from Eq. 3 from Sect. 3.2. The difference between Fig. 4a and 4b, then, is whether blocking is included when considering in-transit costs.

A final point from Fig. 4a is the trend as concurrency increases—in-line visualization increases at a much higher rate than transfer costs. Consider the example of isosurfacing, with $Alloc(50\%)$ and $Delay(0)$ i.e., the blue lines on the right of Fig. 4a with triangle glyphs. These experiments have in-line costs that go from 0.6X of the simulation cycle time at the smallest scale to 2.2X for the largest



(a) In this plot, blocking is not considered. Some glyphs above the line are hollow however due to $VCEF$ being insufficient to achieve overall cost savings.



(b) Updating Figure 4a to include blocking costs. This plot demonstrates that our cost model is able to perfectly infer when cost savings can be achieved with in-transit, as only solid glyphs appear above the line.

Fig. 4. Plot of cost savings feasibility for each test case. Each glyph denotes the in-line cost as a function of transfer and resource costs. Glyph size represents the number of simulation nodes used in each test (isosurfacing: 8-1024; volume rendering: 8-2048). Hollow glyphs indicate in-line was more cost efficient and solid glyphs indicate that in-transit was more cost efficient. The black line marks where in-line and in-transit costs are equal. Above the line is where in-transit can be cost effective.

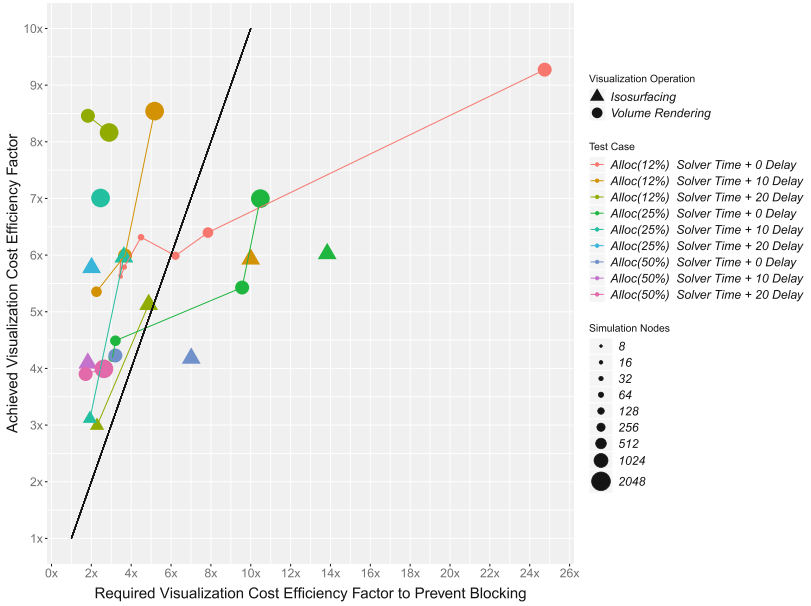


Fig. 5. This plot takes the points from experiments in Fig. 4b where in-transit was cost effective and plots the achieved $VCEF$ as a function of the required $VCEF$ to prevent blocking. The black line is Eq. 8. Points above the line did not block, while those below did block. This plot shows two things: first, the necessary $VCEF$ speedup required to prevent blocking, and second, that cost feasibility is possible even with simulation blocking.

scale. Further, the x-values (i.e., transfer cost and resource cost) change in a much more modest way (0.75X to 0.85X, with this representing only a variation in transfer since the resource cost is fixed at 0.5 for this case). This is a critical point to bring up for in-line visualization: It can be very difficult to scale some algorithms up to the scale of the simulation without incurring huge penalties. All of the other families of experiments exhibit a similar trend, with little variation in X (transfer and resource) and significant increases in Y (in-line visualization) as scale increases. Extrapolating forward, the opportunities demonstrated in our experiments will only become greater as supercomputers get larger and larger.

5.3 Achieved Cost Savings

Figure 5 extends Fig. 4b by plotting the results of Eq. 8 for each of the points that did provide cost savings. Equation 8 calculates the required $VCEF$ value for a in-transit experiment to not block the simulation. While blocking the simulation is certainly not an ideal configuration, it is still possible to achieve cost savings if the cost savings gained through $VCEF$ is greater than the cost of the blocked simulation. About a third of the experiments that provided cost savings from Fig. 4b actually blocked the simulation (points to the right of the black line).

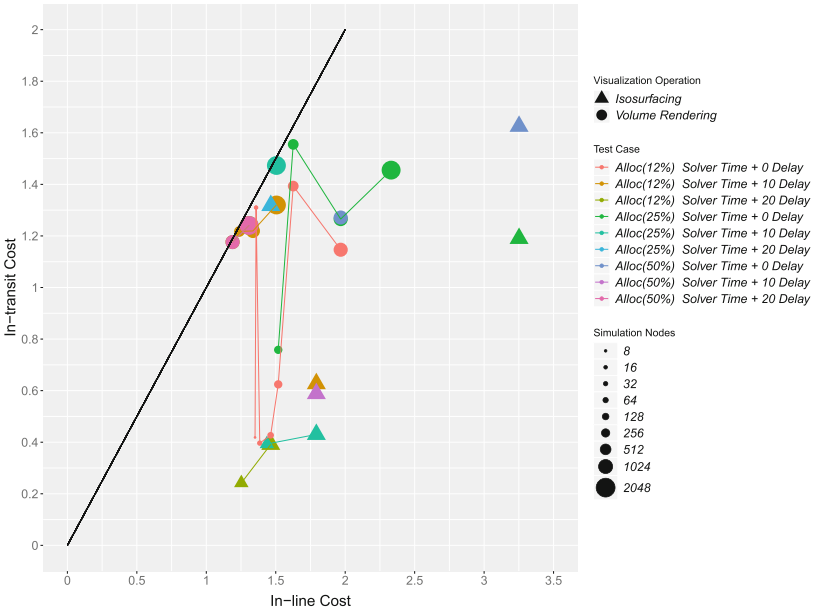


Fig. 6. This plot takes the points from experiments in Fig. 4b where in-transit was cost effective and plots the in-transit cost as a function of the in-line cost using Eq. 9 (if no blocking occurred), or Eq. 10 (otherwise). The black line indicates where costs are equal.

The main takeaway from this plot though, is the rate at which $VCEF$ allowed in-transit visualization to achieve cost savings and prevent blocking. About two thirds of the cases that achieved cost savings did so by not blocking the simulation. This was in large part due to the high values for $VCEF$ that were achieved in those cases.

Looking back to the intuition we established in Sect. 5.1 about which experiments would be viable from a cost savings standpoint, we see that our intuition was correct. Our intuition was that volume rendering would lead to more experiments with cost savings vs. isosurfacing due to its high $VCEF$ values across all concurrencies, whereas isosurfacing only had high $VCEF$ values at high concurrency. Looking at Fig. 5, we see that the majority of the points are for volume rendering, 19 cases were more cost efficient, vs. isosurfacing, 9 cases being more cost efficient. This trend indicates two important things: first, at even higher concurrency we should expect to see larger values for $VCEF$, with even more cases where in-transit is more cost efficient, and second, in future as more algorithms are studied, those with even more communication than volume rendering should see even greater cost savings due to $VCEF$.

Figure 6 takes all of the cases that achieved cost savings from Fig. 4b and shows what the observed in-transit and in-line costs were in each case. The further the points are from the black line the larger the in-transit cost savings.

This chart shows that 30 cases out of a possible 58 cases from Fig. 4a were able to achieve cost savings. Meaning that overall, out of our 153 in-transit tests, we demonstrated high $VCEF$ values and cost savings in 30, or 20%, of our cases. We note that these test cases were originally conceived for a study on the fastest time to solution, not cost savings, so seeing 20% of cases costing less is encouraging. Stated differently, our experiments did not focus on optimizing over resources, and so it is possible that more success could have been found. By focusing on smaller allocations, these studies should see a much higher percentage of cases where in-transit is the most cost efficient choice.

6 Conclusion

The primary results from this paper are three-fold: (1) VCEF values are surprisingly high, and in particular high enough to create opportunities for in-transit to be cost effective over in-line, (2) a model for considering the relative costs between in-transit and in-line that incorporates $VCEF$, and (3) consideration of that model over a corpus of data that demonstrated that VCEF-based savings do in fact create real opportunities for in-transit cost savings. We feel this result is important, since it provides simulation teams a valuable metric to use in determining which in situ paradigm to select. Combined with in-transit's other benefits (such as fault tolerance), we feel this new information on cost could be impactful in making a decision. In our studies, our communication-heavy algorithm showed more promise for in-transit cost benefit than the computation-heavy algorithm. This observation speaks to an additional role for in-transit: sidestepping scalability issues by offering the ability to run at lower concurrency. This is particularly important as the visualization community considers critical algorithms like particle advection, topology, connected components, and Delaunay tetrahedralization.

The results of this study open up several intriguing directions for future work:

The first direction is in selecting an in-transit allocation that is likely to create cost benefits. Our corpus of data was originally conceived for a study on time savings. This is why it included configurations like $Alloc(50\%)$, which have very little chance of providing cost savings. Saying it another way, although we put little effort into choosing configurations that could achieve cost savings, we still found these cost savings occurred 20% of the time. If we put more effort into choosing such configurations, perhaps by incorporating the work of Malakar [12,13], who had complementary ideas on choosing allocation sizes and analysis frequencies, this proportion could rise significantly. A twin benefit to choosing an appropriately sized in-transit allocation is that potentially more nodes would be available for simulation use, as over allocating an in-transit allocation can limit the maximum size of a simulation scaling run.

The second direction is in further understanding of $VCEF$. For our study, we ran production software for two algorithms. We were able to observe $VCEF$ factors after the run, but we are not able to predict them. Predicting $VCEF$ is hard—it will vary based on algorithm, data size, architecture, and possibly

due to data-dependent factors. However, being able to predict $VCEF$ would have great benefit in being able to choose cost effective configurations. Further, our test configuration had in-line experiments that were restricted to the MPI approach of our studied simulation code (one core per MPI task). While this configuration reflects an advantage of in-transit (i.e., freedom to select the optimal configuration), this likely boosted $VCEF$, and future work should evaluate the extent of $VCEF$ when in-transit runs does not have such an advantage.

The third direction is in considering more alternatives to blocking. Making the choice to block simplified our cost model and study. A twin choice was to ignore idle time—we could have tried to do "more visualization" when the in-transit resources completed their initial task and went idle. Making a system that is more dynamic (not blocking and instead visualizing data from the next time step and/or also adding tasks when there is idle time) would be an interesting future direction. Such a system would be able to realize cost savings compared to in-line, provided $VCEF$ can offset transfer costs.

Acknowledgments. This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan). 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 (LLNL-CONF-805283).

References

1. Cloverleaf3d. http://uk-mac.github.io/CloverLeaf3D/. Accessed 19 Dec 2018
2. Bauer, A.C., et al.: In Situ methods, infrastructures, and applications on high performance computing platforms, a state-of-the-art (STAR) report. In: Proceedings of Eurovis Computer Graphics Forum, vol. 35(3) (2016)
3. Childs, H., et al.: In situ visualization for computational science. IEEE Comput. Graph. Appl. (CG&A) **39**(6), 76–85 (2019)
4. Docan, C., et al.: Dataspaces: an interaction and coordination framework for coupled simulation workflows. Cluster Comput. **15**(2), 163–181 (2012)
5. Friesen, B., et al.: In situ and in-transit analysis of cosmological simulations. Comput. Astrophys. Cosmol. **3**(1), 4 (2016)
6. Kress, J., 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, pp. 1–6. ISAV2015, ACM, New York (2015). https://doi.org/10.1145/2828612.2828623
7. Kress, J., et al.: Comparing the efficiency of in situ visualization paradigms at scale. In: ISC High Performance, Frankfurt, Germany, pp. 99–117, June 2019
8. 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)

9. Liu, Q., et al.: Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. Concurr. Comput. Pract. Exp. **26**(7), 1453–1473 (2014)
10. Ma, K.L.: In situ visualization at extreme scale: challenges and opportunities. IEEE Comput. Graphics Appl. **29**(6), 14–19 (2009)
11. Ma, K.L., et al.: In-situ processing and visualization for ultrascale simulations. In: Journal of Physics: Conference Series, vol. 78, p. 012043. IOP Publishing (2007)
12. Malakar, P., et al.: Optimal scheduling of in-situ analysis for large-scale scientific simulations. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 52. ACM (2015)
13. Malakar, P., et al.: Optimal execution of co-analysis for large-scale molecular dynamics simulations. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 60. IEEE Press (2016)
14. Mallinson, A., et al.: Cloverleaf: preparing hydrodynamics codes for exascale. In: The Cray User Group 2013 (2013)
15. Moreland, K., et al.: VTK-m: accelerating the visualization toolkit for massively threaded architectures. Comput. Graphics Appl. **36**(3), 48–58 (2016)
16. Morozov, D., Lukic, Z.: Master of puppets: cooperative multitasking for in situ processing. In: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing, pp. 285–288. ACM (2016)
17. 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)
18. Zhang, F., et al.: In-memory staging and data-centric task placement for coupled scientific simulation workflows. Concurr. Comput. Pract. Exp. **29**(12), 4147 (2017)