Comparing Time-to-Solution for In Situ Visualization Paradigms at Scale

James Kress*
Oak Ridge National Laboratory
University of Oregon

Matthew Larsen
Lawrence Livermore National Laboratory

Jong Choi
Oak Ridge National Laboratory

Mark Kim
Oak Ridge National Laboratory

Matthew Wolf
Oak Ridge National Laboratory

Norbert Podhorszki
Oak Ridge National Laboratory

Scott Klasky
Oak Ridge National Laboratory

Hank Childs
University of Oregon

David Pugmire
Oak Ridge National Laboratory

1 Introduction

This short paper considers time-to-solution for two in situ visualization paradigms: in-line and in-transit. It is a follow-on work to two previous studies. The first study [13] considered time-to-solution (wall clock time) and total cost (total node seconds incurred) for a single visualization algorithm (isosurfacing). The second study [14] considered only total cost and added a second algorithm (volume rendering). This short paper completes the evaluation, considering time-to-solution for both algorithms. In particular, it extends the first study by adding additional insights from including a second algorithm at larger scale and by doing more extended and formal analysis regarding time-to-solution. Further, it complements the second study as the best in situ configuration to choose can vary when considering time-to-solution over cost. It also makes use of the same data corpus used in the second study, although that data corpus has been refactored with time-to-solution in mind.

While total cost is often a computational scientist's most important goal when considering in situ paradigms, we believe that there are multiple use cases in HPC that motivate time-to-solution. One motivation for this problem includes "urgent HPC," i.e., real-time monitoring and fast turnaround. Examples include weather prediction [18], wildfires [22], hurricanes [12], earthquakes [10], and other catastrophic global events [5]. In these cases, fast in situ visualization helps the overall goal of each simulation. Another motivation is when domain scientists are actively studying the results (urgent HPC or otherwise) and would like to get visualizations as quickly as possible. One important use case within this latter motivation is the combination of simulation, observation, and experiment [9, 27]. Overall, these motivations form the fundamental premise behind this study: that in some cases domain scientists will want in situ visualization results as quickly as possible.

To this end, we present a study comparing time-to-solution for in-line and in-transit in situ visualization, measuring impact on the ability of the simulation to progress quickly. Our contributions from this study inform desirable in situ configurations across a variety of simulation scales for both a computation-bound and communication-bound visualization operation.

2 Related Works

There are many resources that consider motivation, challenges, and solutions for in situ processing, for example [3,6,26], as well as many others which are summarized in our two preceding studies [13, 14].

With respect to this current paper, there are several studies that specifically consider time-to-solution from the perspective of in-transit and in-line. Morozov et al. [24] 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. Friesen et al. [8] describes a setup where in-line and in-transit visualization are used in conjunction with a cosmological code to run two different analysis routines. Bennett et al. [4] use both in-line and in-transit techniques for analysis and visualization of a turbulent combustion code. Ayachit et al. [2] performed a study of the overheads associate with using the generic SENSEI data interface to perform in situ analysis using both in-line and in-transit methods. The common theme between these and other studies is that they primarily consider analysis pipelines, which can have different communication and computation overheads versus visualization pipelines. As such, we expect visualization algorithms to exhibit significantly different patterns than those observed in those works that focused specifically on analysis pipelines.

There are four highly relevant works preciding this work:

- Oldfield et al. [25] consider in-transit and in-line times for analysis tasks, but only see a small margin of cases where in-transit is faster, due to the scaling characteristics of the algorithms they studied.
- Malakar et al. did twin studies on cost models, one for in-line [19] and one for in-transit [20]. These studies did not consider optimizing the time-to-solution. Instead, they considered optimizing analysis frequencies and resource allocations, which is complementary to our effort.
- The authors of this paper considered tradeoffs between in-transit and in-line in two previous works [13, 14]. The first previous study showed strong evidence for in-transit time savings for the simulation. However, the algorithm considered was computation-heavy, so the extent of the effect was smaller. The second study focused analysis exclusively on cost savings, which is complementary to the current paper. Furthermore, the current paper focuses exclusively on time savings of both computation-heavy and communication-heavy visualization algorithms, giving a broader range of insight than previous work.

We approach the problem in a different way from the prior work. First, we concentrate on in situ visualization pipelines. Second, we focus specifically on two types of in situ processing (in-line and in-transit) from the perspective of simulation cycle time, visualization type, resource requirements, and how different combinations of these factors impact the final time-to-solution of the simulation.

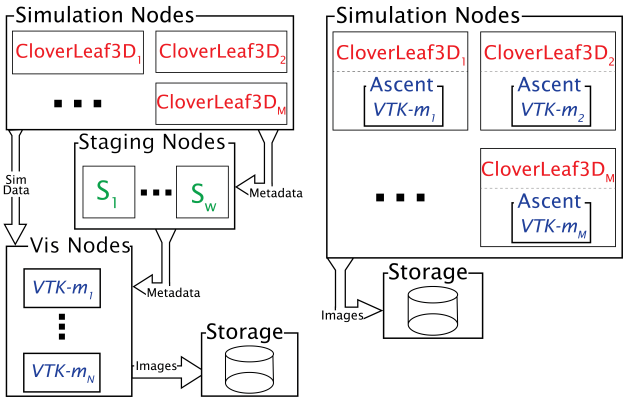
3 Factors Affecting Time-to-Solution

When it comes to minimizing time-to-solution, the primary drawback of incorporating in situ visualization routines into a simulation code are the negative effects on the simulation's runtime. In-line visualization pauses the simulation while the visualization completes.

*e-mail: kressjm@ornl.gov

The impact of this pause varies, ranging from minimal for fast visualization operations to prohibitive for slower, communication-heavy ones. Conversely, in-transit pauses the simulation while the data is being transferred from the simulation nodes to the visualization nodes. This pause can be short or long, depending upon a number of factors. The pause will be shorter if the visualization nodes are ready to receive data as soon as the simulation completes a step and is ready to transmit data. The pause will be longer if the simulation completes a step and the visualization nodes are still busy finishing operations on the previous time step. In this case, often referred to as "blocking," the simulation will have to wait for the visualization nodes to finish, and then transfer the data, incurring a larger time penalty. We note that "blocking" is not the only possible response to simulation being faster than the visualization (they could be ignored instead), but it is the choice we made in the context of this study.

Given that each in situ paradigm necessitates pausing the simulation to some degree, the paradigm that pauses for the least amount of time will minimize time-to-solution. In order for in-line to have the smallest impact on overall runtime, the visualization needs to scale well at the concurrency level of the simulation. In order for in-transit to have the smallest impact on overall runtime, the data transfer needs to be fast, and the visualization needs to scale well at the concurrency level of the smaller in-transit allocation.



(a) The in-transit visualization used in this study. With this mode, the simulation and visualization operate asynchronously, and each have their own dedicated resources.

(b) The in-line visualization used in this study. With this mode, the simulation and visualization alternate in execution, sharing the same resources.

Figure 1: Comparison of the two types of in situ used in this study.

4 Corpus of Data

This section provides a brief overview of the experiments that were used to form our corpus of performance data. These experiments were also used for previous studies [13, 14] (see Introduction for more details), and these studies contain further details about experimental configurations.

In terms of in situ setup, Figure 1 captures our in-transit and in-line configurations. The computational simulation was of hydrodynamics, specifically compressible Euler equations, using the CloverLeaf3D [1, 21] proxy-application. In terms of visualization, Ascent [15] and VTK-m [23] were used for both in-transit and in-line, and the Adaptable I/O System (ADIOS) [17] was used to transport data in the in-transit case, using its RDMA capabilities [7, 28]. We ran Cloverleaf3D with 128^3 cells per process, meaning the total number of cells ranged from 268 million to 68 billion cells. Finally, our tests ran for 100 time steps, and visualization was performed every time step.

Our experiments varied the following factors:

- **Configuration.** We ran five types of configurations:
  - **Sim only:** Baseline simulation with no visualization.
  - **In-line:** Simulation with in-line visualization.
  - **Alloc(12%):** In-transit using an additional 12% resources.
  - **Alloc(25%):** In-transit using an additional 25% resources.
  - **Alloc(50%):** In-transit using an additional 50% resources.
- **Concurrency.** The full concurrency details are in Table 1. One important note however, is that isosurfacing was run up to 16,384 cores, while volume rendering was run up to 32,768 cores. This discrepancy is due to when the experiments were performed, and that Titan was decommissioned before we could run the larger isosurfacing case.
- **Visualization algorithm.** There were two options: isosurfacing (which included parallel rendering of the isosurface Radix-k [11]) and volume rendering (using a VTK-m variant of work by Larsen et al. [16]).
- **Simulation cycle time.** This factor affects whether in-transit is likely to block. We varied the cycle time artificially, by adding a delay via sleep commands. There were three options: "0 Delay" (which was about 5 seconds per cycle), "10 Delay" (about 15 seconds total), and "20 Delay" (about 25 seconds).

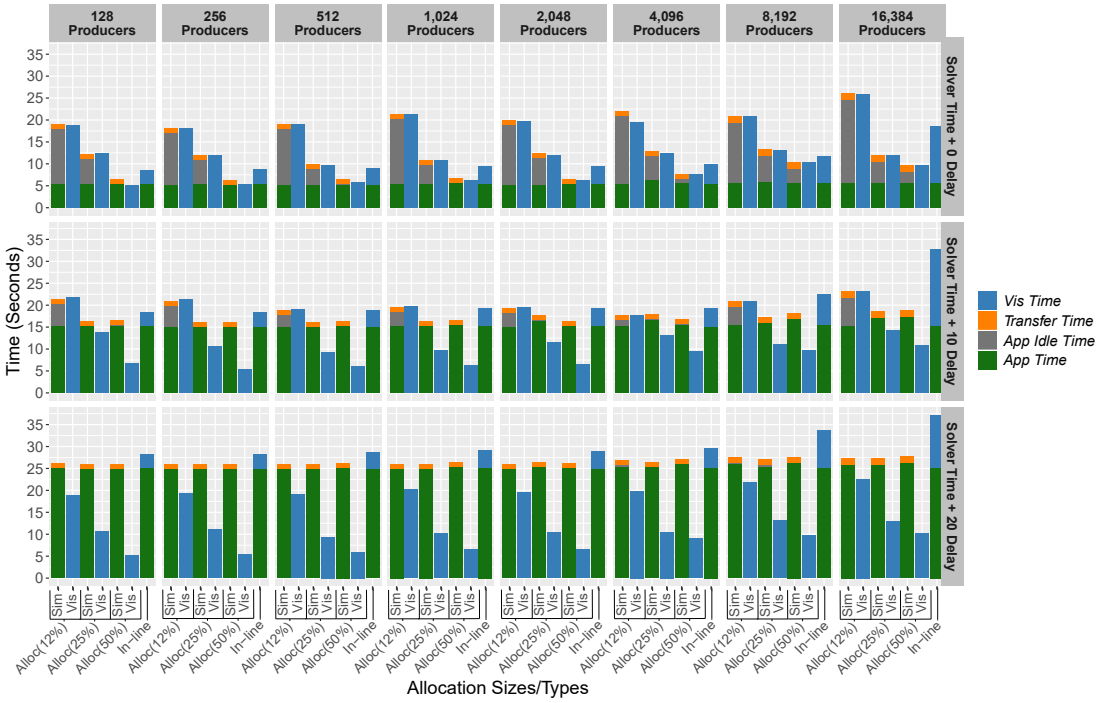
Finally, the experiments were performed on the Titan supercomputer at the Oak Ridge Leadership Compute Facility (OLCF). Since CloverLeaf3D only runs on CPUs, visualization was also performed on the CPUs.

5 Results

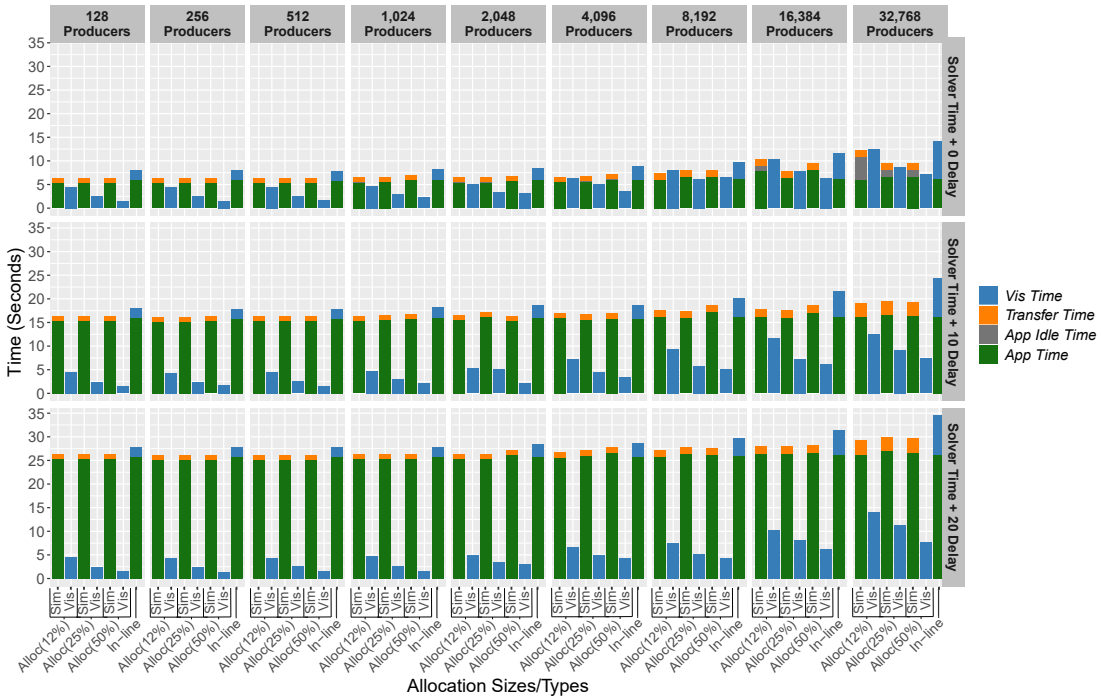
The objective of our study is to understand time-to-solution for both in-transit and in-line in situ for a computation-bound and a communication-bound visualization algorithm. Our results are organized into two parts. First, Section 5.1 analyzes the factors behind performance for both algorithms under both paradigms. Second, Section 5.2 synthesizes the findings by cross comparing the results from our isosurfacing and volume rendering experiments. Lastly, Section 5.3 examines the pitfalls for each in situ paradigm, in particular scalabilibity for in-line and the conditions that cause blocking for in-transit.

Table 1: Resource utilization for each experiment.

Study Configurations	**Sim Cores**	**128**	**256**	**512**	**1024**	**2048**	**4096**	**8192**	**16384**	**32768**
	Sim 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
In-transit *Alloc*(25%)	Vis Nodes	2	4	8	16	32	64	128	256	512
In-transit *Alloc*(50%)	Vis Nodes	4	8	16	32	64	128	256	512	1024



(a) Per time step breakdown for isosurfacing and surface rendering.



(b) Per time step breakdown for volume rendering.

Figure 2: Comparing the total time per step for using in-transit and in-line methods. This chart looks at time from the applications perspective, meaning that the time for in-transit visualization is only how long it takes to transfer the data from the application, unless the in-transit resources block, in which case the application becomes idle. In-transit visualization is broken down into the time it takes to receive data from the application and how long the application is blocked by the in-transit resources being too slow. A second column is present for each in-transit case that shows how long the in-transit resources were active during a single time step, giving a better sense of where blocking and idle time occurs. In-line has a single time, how long it took to perform visualization. In terms of colors, orange bars represent the time the application was active, purple bars the time the application was idle, green bars the time to transfer data to the in-transit resources, and red and blue bars represent the time to perform in-transit visualization and in-line visualization, respectively.

5.1 Analyzing Time-to-Solution

This section analyzes where each algorithm spent their time in the experiments, and draws conclusions about performance under in situ paradigm and system constraints.

All the analyses in this section are based on Figure 2, which shows the total time per time step for our experiments. This figure is divided into two sub-figures, with Figure 2a focusing on the isosurfacing plus rendering experiments and Figure 2b focusing on the volume rendering experiments.

5.1.1 Isosurfacing

With respect to the isosurface algorithm (Figure 2a), in-transit in situ suffered from poor performance in the $Alloc(12\%)$ and $Alloc(25\%)$ experiments at $Delay(0)$. All of these experiments have large portions of time where the simulation is blocked because the in-transit visualization was unable to keep up with the simulation. This blocking effect made it so that in-line in situ was the most performant choice for all scales but the largest (16,384 processes). Another observation from this chart is the absence of simulation blocking in all of the $Delay(20)$ cases. In each of those cases, the only time delay for the simulation was the time to transfer data to the in-transit resources. However, these cases also showcase the other negative of in-transit visualization: idle in-transit resources. In every case, the in-transit resources were idle for some percentage of the simulation cycle, the worst being $Alloc(50\%)$, which was idle for up to 80% of each simulation cycle. This level of idle time means that resources were severely over allocated, and either the resources need to be reduced, or the visualization pipeline needs to be adaptive, i.e., dynamically add new visualization operations in order to make productive use of the resources.

5.1.2 Volume Rendering

With respect to the volume rendering algorithm (Figure 2b), in most cases in-transit volume rendering was faster than the simulation cycle time. $Delay(0)$ caused the simulation to block with only the two largest cases (16,384 and 32,768 processes). This change from isosurfacing is because volume rendering is communication bound, and is more efficient at smaller scale. This led in-transit volume rendering to be faster than in-line in every single experiment we performed. This characteristic (not blocking the simulation) also had a pitfall, however, which was idle in-transit resources. As evidenced by the large idle times (up to 88% of the total runtime), some of the allocations were too large. Similar time-to-solution could have been achieved by using fewer resources, which would have reduced the resource idle time.

5.2 Synthesis Across Algorithms

This section synthesizes findings from Sections 5.1.1 and 5.1.2. It also analyzes the differences between algorithms and in situ paradigms to provide general guidelines for which in situ method will perform the best with a given workload and concurrency. The main finding is that there are very few cases where in-line is faster. This is especially apparent in the volume rendering tests, where in-transit was faster in every case. That said, there are cases for isosurfacing where in-line was the fastest choice, namely the cases with fast simulation cycle time.

Looking more in depth at the time charts (Figure 2), there are marked differences between the performance of the isosurfacing and volume rendering runs. The isosurfacing tests have large periods of blocking in the $Delay(0)$ cases, seen in the figure as *App Idle Cost*, whereas the volume rendering runs have very little. This observation further highlights the need to understand performance of visualization algorithms at different levels of concurrency, as the blocking time was cut by more than 50% in almost all cases when the in-transit resources were doubled from $Alloc(12\%)$ to $Alloc(25\%)$.

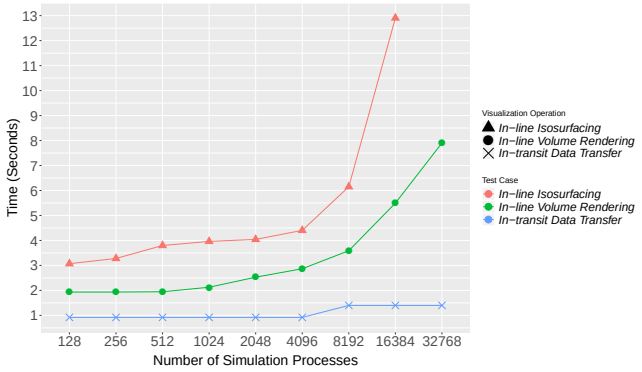


Figure 3: Total time per step for in-transit in situ to transfer data off of the simulation nodes compared against the time that in-line takes to perform either the isosurfacing or volume rendering operations. In essence, this chart shows how long the simulation had to pause each simulation step for visualization to take place, either in-line, or in-transit by moving the data to a separate allocation and performing the visualization asynchronously to the simulation.

Figure 3 highlights the differences in time to perform an in-transit data transfer vs. the time it takes to perform in-line visualization for both isosurfacing and volume rendering. In this figure there are no instances where the data transfer takes more time than the associated in-line visualization operation. This means that, for the algorithms tested, in-transit visualization always has a chance to be faster than in-line visualization. Another interesting trend in this figure is the widening gap between data transfer time and the comparable in-line visualization times as scale increases. This indicates that in-transit will have even more advantages as scale increases.

5.3 Pitfalls for Each In Situ Paradigm

This section highlights important pitfalls that exist for both in-line (5.3.1) and in-transit (5.3.2) in situ. These pitfalls vary in intensity based on visualization algorithm and computational scale.

5.3.1 Poor In-line Scalability

The primary pitfall with in-line in situ is the poor scaling performance of visualization algorithms. This behavior is highlighted in Figure 4 which gives the cumulative rendering and compositing

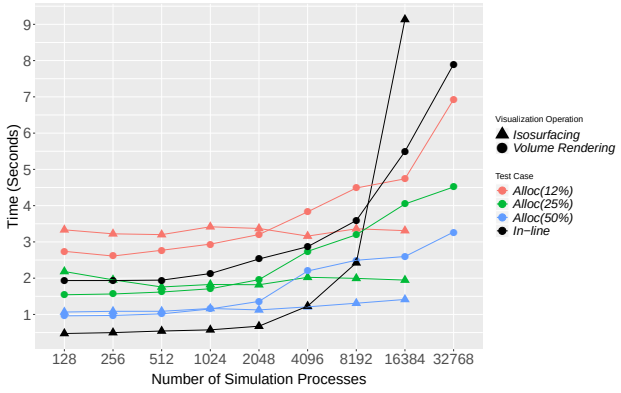


Figure 4: Total time per step to render and composite an image, both in-transit and in-line. The results from isosurfacing (triangles) and volume rendering (circles) are shown. Experiments are grouped by color (configuration) and connected by lines (concurrency sequence).

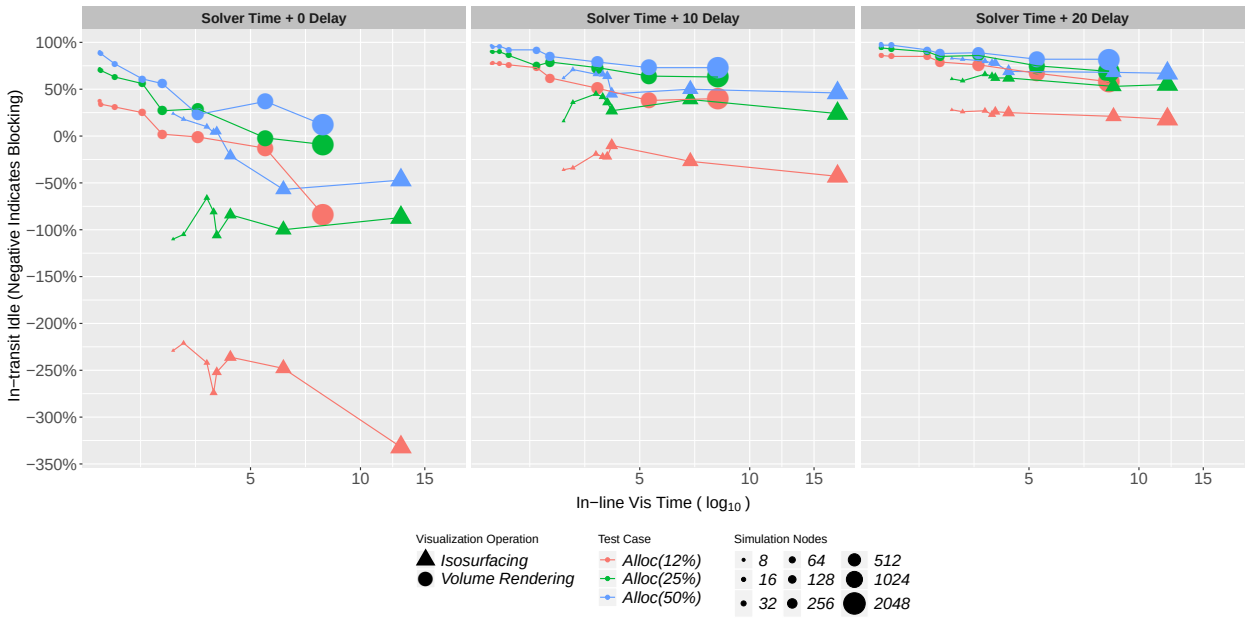


Figure 5: An analysis of whether in-transit resources are idle or blocking during the course of the simulation. The y-axis shows how idle the in-transit resources were during each simulation cycle. An idle time of 0% indicates perfect harmony — in-transit resources were always busy and never blocked. 100% idle means the in-transit resources were always idle, while -100% idle means that the in-transit resources blocked the simulation from proceeding for an entire simulation cycle. The results from in-transit isosurfacing (triangles) and volume rendering (circles) are shown. Each glyph is scaled by the concurrency of the experiment (isosurfacing: 8-1024; volume rendering: 8-2048). Experiments are grouped by color (configuration) and connected by lines (concurrency sequence).

times for both in-line and in-transit in situ. Rendering scales very well for both in-line and in-transit up through $4,096$ processes. Beyond that, the communication at higher levels of concurrency leads to a drop in scalability for in-line isosurfacing and volume rendering. Isosurfacing for example, has the compositing and rendering time rise from 1 second per step at $4,096$ processes up to 9 seconds at $16,384$ processes, an increase of *9x*. Volume rendering has a much smaller rise in compositing and rendering time, from 2.5 seconds at $4,096$ processes, up to 5.5 seconds at $16,384$ processes, an increase of about *2.5x*. Overall, these trends show how in-line suffers from scalability problems. Further, since in-transit can quickly transfer data at large concurrencies (See Figure 3), the cumulative time of data transfer and rendering is much less than the comparable in-line runs.

5.3.2 Can In-transit Visualization Keep Up?

The primary pitfall of in-transit in situ is whether or not it is able to keep up with the cycle time of the simulation, while at the same time not wasting compute resources. This pitfall is affected by many factors: algorithmic intensity, inter-process communication, scale of the in-transit resources, and the cycle time of the simulation. This complex mix shows why in-transit can be hard to optimize, as slight misconfigurations can lead to drastic changes in performance.

Figure 5 highlights this feature of in-transit visualization, showing that it is difficult to keep the in-transit resources busy for an entire simulation cycle while not blocking the simulation. This figure shows how long the in-transit resources were idle each simulation cycle in relation to the length of a simulation cycle. For example, in order for the in-transit idle percentage to be 0%, the simulation and visualization cycle time would need to be the same. If the in-transit idle percentage is -100% idle, the visualization time would be double the simulation cycle time, leading to significant delays.

Looking at the $Delay(0)$ column of Figure 5, there is a large variation in the idle times for in-transit. In the worst case (from the simulations perspective) in-transit blocks the simulation from proceeding for 3.5 (-350%) simulation cycles. While the best cases have in-transit being neither idle, nor blocking the simulation. This trend highlights the effects of simulation cycle time on the size of the in-transit resources necessary to complete a task in time. Looking at the $Delay(10)$ and $Delay(20)$ columns, almost every case was able to complete without blocking the simulation. Overall, the volume rendering runs were less adversely affected by simulation cycle time, further highlighting the performance differences between compute and communication bound visualization algorithms.

6 Conclusion

This short paper presents a study that compares the time-to-solution for in-line and in-transit in situ visualization, and provide an analysis of when and why one paradigm is faster than another. Our hypotheses entering this work was that there should be clear distinctions between scenarios in which an algorithm performs well in-line or in-transit, and the major contribution of this work is confirmation of that hypothesis. From these experiments we found that our communication-bound workload ran faster in-transit versus in-line, in all cases. In addition, we found that our computation-bound workload was faster in-line in many cases with a short simulation cycle time; however, as simulation cycle time increased, in-transit became faster. These findings inform desirable in situ configurations that can help create performant workflows. Further contributions of this work include additional analysis of when to choose in-line or in-transit in situ by comparing algorithm performance across a variety of simulation cycle times and in-line and in-transit allocation sizes.

We feel there are three interesting areas of future work. First, this study explored two different types of algorithms. We intend to study additional classes of algorithms to understand their behavior and performance at scale using different in situ paradigms. Second, while trends at higher concurrencies are clear, additional runs at

even larger scale could be beneficial. Timings started to change significantly for in-line at the largest scales. Examination of these trends at higher scales will provide additional insight into in situ visualization. Finally, the *Alloc* sizes chosen in this study were much too large for some of our experiments. Studying lower percentages of in-transit allocations will help to reduce resource requirements for future in situ integrations, while also ensuring a fast time-to-solution.

REFERENCES

[1] Cloverleaf3d. `http://uk-mac.github.io/CloverLeaf3D/`. Accessed: 2018-12-19.

[2] U. Ayachit 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.

[3] A. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, et al. In situ methods, infrastructures, and applications on high performance computing platforms. In *Computer Graphics Forum*, vol. 35, pp. 577–597. Wiley Online Library, 2016.

[4] J. C. Bennett 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.

[5] N. Brown, R. Nash, G. Gibb, B. Prodan, M. Kontak, V. Olshevsky, and W. Der Chien. The role of interactive super-computing in using hpc for urgent decision making. In *International Conference on High Performance Computing*, pp. 528–540. Springer, 2019.

[6] H. Childs et al. In Situ Visualization for Computational Science. *IEEE Computer Graphics and Applications (CG&A)*, 39(6):76–85, Nov./Dec. 2019.

[7] C. Docan, M. Parashar, and S. Klasky. Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing*, 15(2):163–181, 2012.

[8] B. Friesen et al. In situ and in-transit analysis of cosmological simulations. *Computational Astrophysics and Cosmology*, 3(1):4, 2016.

[9] G. Gibb, R. Nash, N. Brown, and B. Prodan. The technologies required for fusing hpc and real-time data to support urgent computing. In *2019 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*, pp. 24–34, Nov 2019. doi: 10.1109/UrgentHPC49580.2019.00009

[10] S. Habata, M. Yokokawa, S. Kitawaki, et al. The earth simulator system. *NEC Research and Development*, 44(1):21–26, 2003.

[11] W. Kendall, T. Peterka, J. Huang, H.-W. Shen, and R. Ross. Accelerating and benchmarking radix-k image compositing at large scale. In *Proceedings of the 10th Eurographics conference on Parallel Graphics and Visualization*, pp. 101–110. Eurographics Association, 2010.

[12] M. Kontak, J. Vidal, and J. Tierny. Statistical parameter selection for clustering persistence diagrams. In *2019 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*, pp. 7–12, Nov 2019. doi: 10.1109/UrgentHPC49580.2019.00007

[13] J. Kress et al. Comparing the Efficiency of In Situ Visualization Paradigms at Scale. In *ISC High Performance*, pp. 99–117. Frankfurt, Germany, June 2019.

[14] J. Kress et al. Opportunities for cost savings with in-transit visualization. In *ISC High Performance 2020*. ISC, Frankfurt, Germany, 2020.

[15] M. Larsen 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.

[16] M. Larsen, S. Labasan, P. Navrátil, J. Meredith, and H. Childs. Volume Rendering Via Data-Parallel Primitives. In *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)*, pp. 53–62. Cagliari, Italy, May 2015.

[17] Q. Liu 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.

[18] F. Lvholt, S. Lorito, J. Macias, M. Volpe, J. Selva, and S. Gibbons. Urgent tsunami computing. In *2019 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*, pp. 45–50, Nov 2019. doi: 10.1109/UrgentHPC49580.2019.00011

[19] P. Malakar 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.

[20] P. Malakar 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.

[21] A. Mallinson et al. Cloverleaf: Preparing hydrodynamics codes for exascale. *The Cray User Group*, 2013, 2013.

[22] J. Mandel, M. Vejmelka, A. Kochanski, A. Farguell, J. Haley, D. Mallia, and K. Hilburn. An interactive data-driven hpc system for forecasting weather, wildland fire, and smoke. In *2019 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*, pp. 35–44, Nov 2019. doi: 10.1109/UrgentHPC49580.2019.00010

[23] K. Moreland et al. VTK-m: Accelerating the Visualization ToolKit for Massively Threaded Architectures. *Computer Graphics & Applications*, 36(3):48–58, 2016.

[24] D. Morozov and Z. Lukic. 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.

[25] R. A. Oldfield, K. Moreland, N. Fabian, and D. Rogers. 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.

[26] T. Peterka, D. Bard, J. C. Bennett, E. W. Bethel, R. A. Oldfield, L. Pouchard, C. Sweeney, and M. Wolf. Priority research directions for in situ data management: Enabling scientific discovery from diverse data sources. *The International Journal of High Performance Computing Applications*, Mar. 2020. `https://doi.org/10.1177/1094342020913628`. doi: 10.1177/1094342020913628

[27] G. Shipman, S. Campbell, D. Dillow, M. Doucet, J. Kohl, G. Granroth, R. Miller, D. Stansberry, T. Proffen, and R. Taylor. Accelerating data acquisition, reduction, and analysis at the spallation neutron source. In *2014 IEEE 10th International Conference on e-Science*, vol. 1, pp. 223–230, Oct 2014. doi: 10.1109/eScience.2014.31

[28] F. Zhang et al. In-memory staging and data-centric task placement for coupled scientific simulation workflows. *Concurrency and Computation: Practice and Experience*, 29(12), 2017.