Scalable In-Situ Visualization for Extreme-Scale SPH Simulations

Yiqing Zhu
University of Basel
Basel, Switzerland

Osman Seckin Simsek
University of Basel
Basel, Switzerland

Jean M. Favre
Swiss National Supercomputing Centre
Lugano, Switzerland

Rubén Cabezón
University of Basel
Basel, Switzerland

Florina M. Ciorba
University of Basel
Basel, Switzerland

***Abstract*—Large-scale scientific simulations present significant challenges in data processing efficiency. This paper addresses the critical issue of I/O and data processing performance bottlenecks within the domain of extreme-scale Smoothed-particle Hydrodynamics (SPH) and gravity simulations. We present a novel I/O software architecture implemented in the scalable SPH-EXA framework [1], incorporating a variety of in-situ and post-hoc data analysis pipelines, facilitating rapid analysis and visualization of extreme-scale physical datasets. The performance of our I/O architecture is evaluated through comprehensive benchmarking across a wide range of data scales, conducted on the Piz Daint supercomputer [2].**

***Index Terms*—in-situ visualization, HPC, extreme-scale, SPH**

I. Introduction

The rapid growth in supercomputer data transfer capabilities has been remarkable, but meeting the increasing demands of data transfer and processing for scientific simulations remains challenging. Especially during extreme-scale simulations, I/O performance often becomes a critical bottleneck. Furthermore, data processing demands have become more diverse, with a shift towards in-situ and in-transit methodologies [3], [4]. The challenge lies in efficiently processing vast volumes of data, providing domain-scientists with timely and meaningful analysis. Moreover, to better connect analysis and simulations, real-time interaction can sometimes offer a more coherent and insightful exploration [5], [6]. Especially in extreme-scale simulations, monitoring and controlling specific physical parameters is crucial, thus fast collection and analysis of large-scale datasets become critical.

Compression can be a possible solution to meet these requirements. Historically used to reduce data size, compression now applies to simulations as well. In simulations, lossless compression is used for checkpointing to ensure no precision loss happens. Lossy compressors are preferred when data fidelity is not the primary focus. Until recently, compression methods in extreme-scale simulations were underexplored, because the advantage of integrating compressors into the data analysis pipeline becomes apparent only when I/O is the bottleneck of such simulations. Reaching exascale era, the demand for high-performance data analysis and I/O becomes dominant. The need for an architecture that incorporates such functionality thus becomes crucial.

In this work, we explore extreme-scale I/O in SPH simulations using SPH-EXA framework, which generates spatially distributed particle data for various analyses. SPH-EXA, designed for extreme-scale smoothed particle hydrodynamics simulations, features efficient scaling, compatibility across architectures, and a sophisticated data analysis pipeline, such as probability density function (PDF) generation, and comprehensive visualization tools.

We have implemented an I/O pipeline in SPH-EXA and conducted benchmarking experiments with compressors enabled. Our findings offer valuable insights for architects designing high-performance I/O and data analysis infrastructures in particle-based simulations.

II. Background and Related Work

A. In-Situ Data Analysis Tools

An in-situ data analysis pipeline performs data analysis within the same computational node as the simulation, minimizing data transfers as well as allowing real-time analysis. In scientific simulations, in-situ architectures like Ascent [7], and ParaView [8] are widely used, providing immediate insights without the delays of post-hoc analysis.

Ascent is a lightweight in-situ framework for data analysis and visualization with integrated renderers. It supports efficient and extensible data analysis pipelines using CUDA, OpenMP and MPI. Ascent allows customization via YAML files and on-the-fly data processing through the Conduit API [9]. Furthermore, it doesn't have a GUI and can integrate easily into HPC environments.

ParaView/Catalyst is a toolset of in-situ visualization where users can define visualization pipelines with ParaView GUI and connect simulation codes with Catalyst. ParaView is a post-hoc visualization tool from the joint effort of Kitware and LANL [10], [11], and based on that, the in-situ component Catalyst was later developed. Latest Catalyst2 API also uses Conduit [9] for describing data structures and storing parameters. The extended ParaView/Catalyst toolset runs the visualization pipeline in parallel on the same compute nodes as the simulation. They contain MPI-parallelized data analysis and visualization algorithms processed within a supercomputing cluster, with the main process collecting results for final analyses.

Data analysis of a high-performance SPH framework like SPH-EXA requires a lightweight architecture, easy to deploy, portable across multiple architectures, and capable of handling large data volumes with high parallelism. It should also be flexible enough to integrate with various custom data pipelines and offer interactivity for real-time data inspection by domain-scientists. We chose Ascent for our implementation due to its high customizability and robust performance.

B. *Compression*

Compressors are typically applied after data generation to reduce data transfer between hardware, but as data sizes grow in the Exascale era, compact data size becomes crucial for intra-node transfer as well. In the context of in-situ data operations, compression can occur directly in host and GPU memory, optimizing data handling across different phases. Achieving in-memory compression requires compressors to have direct access to simulation data, with an efficient description of the dataset. This access should facilitate the integration of compression techniques into the data processing pipeline, enhancing overall efficiency and effectiveness.

Lossless Compression. Lossless compressors originated from the goal of creating an encoder that achieves the minimum code length required for encoding data without losing information [12]. Modern lossless compressors can be up to 90 times faster than the original Zlib (DEFLATE) algorithm [13], while still approaching its theoretical compression limits.

They are primarily used where data fidelity is fundamental, such as in checkpointing procedures, ensuring the simulation restarts from the original dataset without errors.

Lossy Compression. Lossy compressors reduce data size by prioritizing size over precision, sacrificing some accuracy. Commonly used compressors like SZ [14], ZFP [15], and MGARD+ [16] each have unique error criteria, with ZFP and MGARD optimized for multidimensional, spatial data. Most lossy compressors support parallelization to speed up compression, including GPU-based versions like CUDA for SZ and ZFP, and HIP for ZFP [17]. However, they are not suitable for all cases, as simple error criteria may not capture complex physical details. Accurate analysis requires validating data correctness with domain-specific methods.

C. *Parallel I/O Architecture*

For large-scale scientific simulations, I/O operations can heavily burden resources. Maintaining parallel and asynchronous I/O is crucial to prevent bottlenecks and ensure smooth data analysis. Among the widely adopted I/O technologies specially designed for HPC clusters are: HDF5 [18] and ADIOS2 [19], tailored for their unique storage systems.

HDF5 is a widely used file format in scientific computing, known for handling vast and complex data. It integrates popular compressors like Szip [20] for lossless and SZ for lossy compression. Recent HDF5 updates [21] include Virtual Object Layer (VOL) [22] features, which enable asynchronous I/O and optimize performance in data-intensive workflows.

ADIOS2 is an I/O framework for HPC environments, offering a unified API for both in-memory and external file storage of large datasets. It uses in-memory communication within nodes and MPI-based inter-node communication for parallel I/O operations. ADIOS2 also supports asynchronous I/O, ensuring simulations are not delayed by data operations, and integrates various compressors. Moreover, it supports various file formats like HDF5 and Dataman, enhancing interoperability and integration with existing data storage infrastructures.

Various architectures have been proposed to enhance data analysis performance in HPC environments. Dorier et al. [23] introduced an in-situ analysis architecture that enables elastic visualization through integration with ParaView Catalyst and Ascent, but it lacks extreme-scale experimental results and does not address the I/O subsystem bottleneck. Ravi et al. [4] proposed Runaway, which supports in-transit data compression with a focus on large-scale I/O options, though it is not integrated with data analysis. For extreme

Among the options, HDF5 has a long history and a large user base. However for SPH-EXA, which handles various data formats and compression methods and requires robust, native non-blocking I/O for large-scale data analysis and checkpointing, the I/O foundation must also integrate easily into various pipelines across different programming languages. We chose ADIOS2 for our I/O architecture because it is highly extensible and provides excellent performance across various architectures.

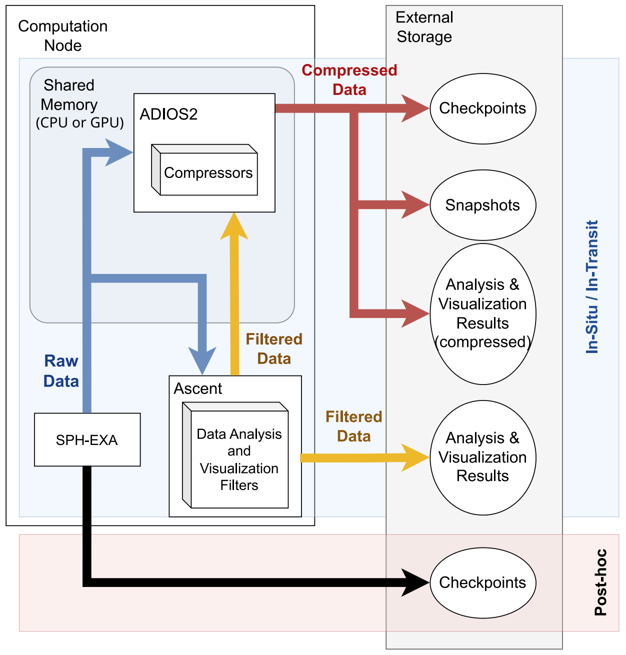
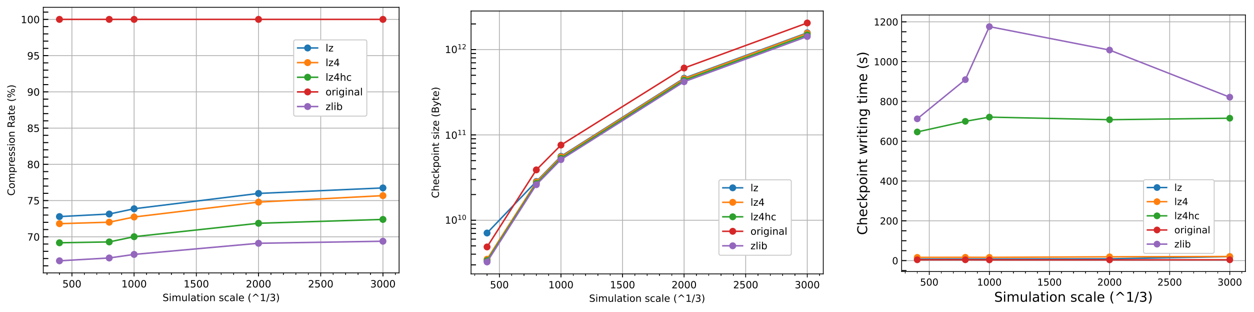


Fig. 1. Data analysis and visualization architecture of SPH-EXA.

III. Workflow Architectural Design

The SPH-EXA design of data analysis and I/O architecture is motivated by the need for efficient data management and real-time analysis in high-performance simulations. With the data I/O API, users can easily define and manage their data fields, ensuring seamless analysis of simulation states. The integration of Ascent for in-situ data analysis allows immediate insights during simulations, reducing the need for



(a) (b) (c)

Fig. 2. Lossless compression of SPH-EXA checkpoints: (a) Compression rates, (b) Checkpoint sizes, (c) Checkpoint writing time of different compressors.

post-processing. The custom API connecting Ascent directly to the ADIOS binary-pack (BP) format enhances efficiency by eliminating intermediate steps. Overall, SPH-EXA aims to streamline simulation workflows and maximize computational efficiency for researchers. The proposed data analysis and visualization architecture is illustrated in Figure 1.

A. Checkpoints and Snapshots

SPH-EXA provides a unified API for simulation checkpointing and data I/O. During runtime, users define the data fields attached to fluid elements (i.e. SPH particles) they wish to export and specify the frequency within SPH-EXA. If no specific output field is defined, SPH-EXA exports a **checkpoint** which can then be used to restart the simulation. **Snapshots** retain specific portions of the simulation data as specified by the user which can then be used for further analysis. Both for checkpoints and snapshots, the SPH-EXA output process is structured as follows: 1). Initializing the file within the file system. 2). Adding file attributes: adding metadata essential for the entire simulation process or for parsing the file. 3). Opening a step. 4). Writing step attributes that are associated with the step only. 5). Writing individual data fields associated with the step. 6). Closing the step handle. 7). Flushing the file buffer and closing the file.

During checkpoint loading, SPH-EXA initially reads metadata from file attributes, then it distributes the datasets to each MPI rank in accordance with the metadata obtained.

B. In-situ Data Analysis

For adaptable in-situ data analysis, we use Ascent as our foundation. Leveraging Conduit with Ascent facilitates the retrieval of hierarchical simulation data from shared memory, enabling further analysis via ADIOS2. By compiling Ascent with OCCA and Raja, we establish a robust infrastructure for high-performance parallel data analysis with MPI.

Instead of using Fides [24] to convert ADIOS2 data to VTK-m, we developed a direct API between Ascent and the ADIOS2 BP format. This allows ADIOS2 to transmit compressed datasets directly to Ascent for decompression. Additionally, we extended Ascent to include ADIOS2 output functionality, enabling SPH-EXA to pre-process and export datasets for post-hoc analysis.

C. Post-hoc Analysis

Post-hoc data analysis involves importing snapshots and checkpoints into offline analysis and visualization tools such as Python scripts, Blender, and VTK. ADIOS2 provides built-in Python support for processing compressed ADIOS2 data, but users can also opt to output data in HDF5 format, which is widely supported by most mainstream tools such as Matlab [25] and openFOAM [26].

IV. Performance Evaluation

A. Experimental Setup

We evaluated compression effects on checkpoint performance and size, with SPH-EXA on simulations on turbulent flows on the XC50 Compute Nodes of Piz Daint [2]. Each Piz Daint node has an Intel Xeon E5-2690 v3 (12 cores, 64 GB RAM) and a NVIDIA Tesla P100 GPU (16 GB RAM). GPUs handle the main SPH computational tasks, while CPUs manage data compression and I/O. Since the GPU-based implementation of lossless compressors are not supported by ADIOS2 yet, we chose to run lossless compressors on CPUs, while lossy compressors (SZ, ZFP) run on GPUs.

Simulation experiments data sizes range from 400^3 to 3000^3 particles, with 14 exported data fields for both snapshots and checkpoints. Experiments are also evaluated in both strong scaling and weak scaling scenarios.

Weak Scaling: We maintained 64 million particles per node across simulation scales of 400^3, 800^3, 1000^3, 2000^3, 3000^3, using 1, 8, 16, 128, and 432 nodes respectively. For 3000^3, each node processed 62.5 million particles.

Strong Scaling: We fixed the simulation size at 1000^3 and varied the number of nodes (16, 20, 24, 28, 32). In both scaling scenarios, each GPU was assigned 1 MPI rank, consistent with the rule of thumb in GPU-centric simulations.

B. Lossless Compression Results

The benchmarked lossless compressors include Zlib [27], LZ, LZ4 [13], and LZ4HC [13]. For these lossless compressors, we opted for the maximum compression level of 9 since empirical experiments indicate the performance improvement by increasing the compression level is negligible with respect to the compressed file size.

We first evaluate the compression rates of various compressors (Figure 2(a)), with compression rate as $C = \frac{\text{size of compressed data}}{\text{size of raw data}}$. The baseline indicated by *original* denotes an uncompressed checkpoint. Zlib achieves the lowest compression rates below 70%, while LZ ranges from 73% to 76%. The differences among these methods are notable but not significant compared to the baseline. Compression efficiency slightly increases with larger simulation scales due to more compression possibilities. Lossless compressors reduce file size by at least 23%, with Zlib performing best, especially at larger scales (Figure 2(b)).

Regarding checkpoint compression, both file size and overhead matter. Figure 2(c) shows that Zlib and LZ4HC incur higher write time overheads.

Zlib offers higher compression ratios, resulting in smaller checkpoint sizes, but it also increases write overhead. On the other hand, LZ4 reduces checkpoint size by up to 27.27% with a 4.23x overhead increase compared to the original ($7.37s$ more when running on 28 nodes), making it a viable option.

Moreover, checkpoint writing shows high scalability. While Zlib's overhead increases, other compressors maintain consistent I/O overhead as simulation scales up. This stability indicates our I/O solution's effectiveness in parallelized writing, with minimal overhead even when multiple nodes write simultaneously even with fast storage writing like LZ and LZ4 ($\sim$1.5 TB in 20 seconds).

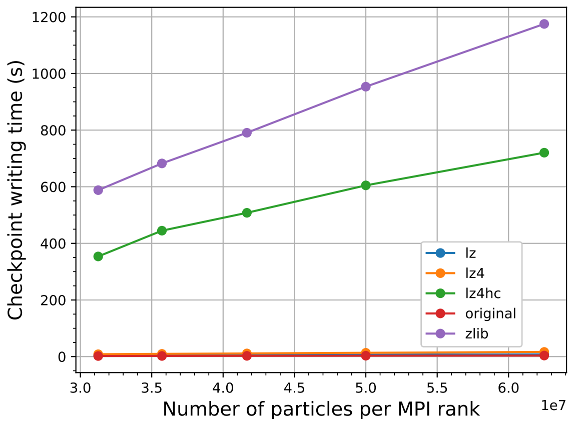


Fig. 3. Comparison of the checkpoint writing time using lossless compression with the strong scaling of the simulation size (lower is better).

In strong scaling benchmarks, we reduced the workload per node while keeping the total workload constant. Again we observed that checkpoint writing time is proportional to the workload on each MPI rank (Figure 3). Compression ratios were unaffected by local workload, as they depend more on the values being compressed than on data size.

C. Lossy Compression Results

We benchmarked lossy compressors cuSZ [28] and cuZFP [15] with error bounds of 10^{-3}, 10^{-5}, and 10^{-7} on double-precision datasets. Lossy compression was used for snapshots, while lossless compression was reserved for checkpointing.

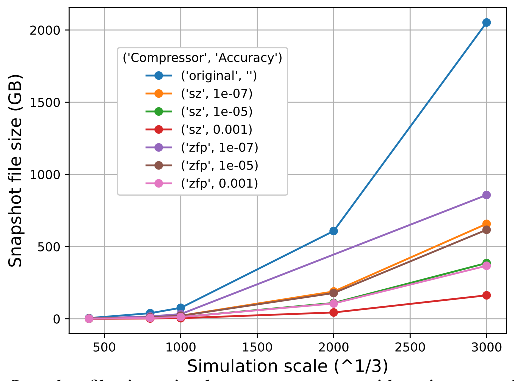


Fig. 4. Snapshot file size using lossy compressors, with various error bounds.

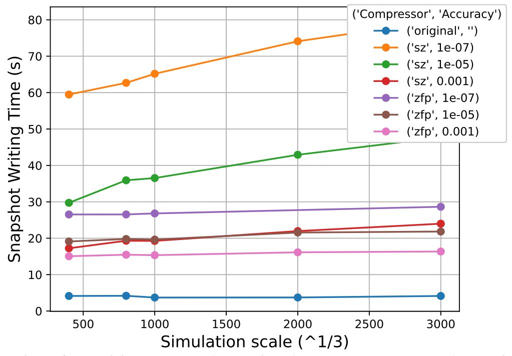


Fig. 5. Time for writing a snapshot using lossy compressors (lower is better), with various error bounds.

Figure 4 shows that both SZ and ZFP reduce snapshot sizes effectively. SZ achieves up to 68% reduction for 3000^3 simulations at 10^{-7} accuracy, and 92% at 10^{-3}. For 1000^3 simulations, ZFP achieves a 57% reduction at 10^{-7} accuracy, and 84% at 10^{-3}. Note that ZFP results are not available for 10^{-7} accuracy due to code limitations.

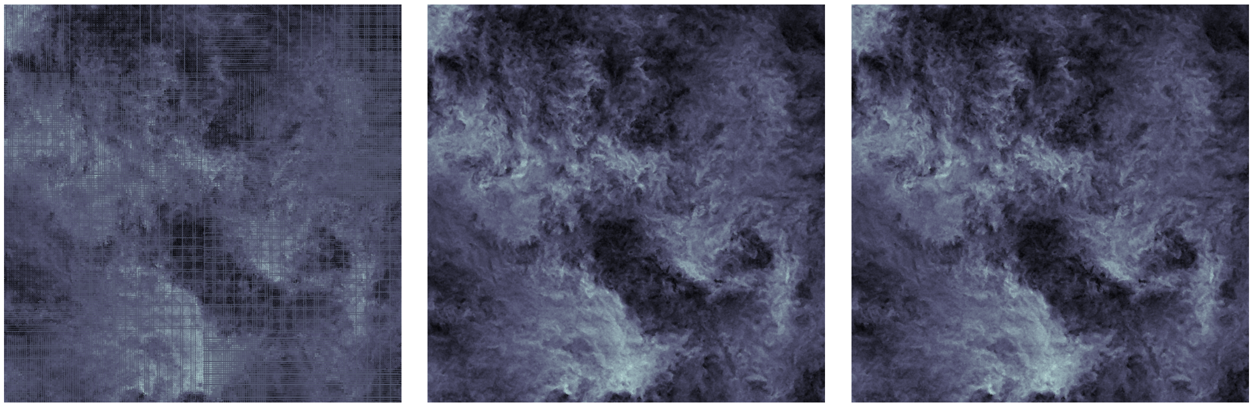
Figure 5 shows snapshot writing times for different lossy compression methods and error bounds. The non-compressed snapshot takes up to 5 seconds as a baseline. Lower accuracy compression reduces writing time, but even at the highest accuracy (10^{-7}), the overhead increase is minimal. For SZ with 3000^3 simulations, the snapshot writing time of under 80 seconds is negligible considering these simulations run for days and months with the benefit of snapshot reduction from 2052 GB to 657 GBs.

These results highlight the benefits of snapshot size reduction, though the impact on accuracy must be considered for simulation correctness and analysis. The effect of accuracy reduction for the visualization is further discussed in IV-D.

D. Data Analysis and Visualization Pipelines

We evaluated lossy compressors in various scenarios for subsonic turbulence analysis and snapshots during simulations:

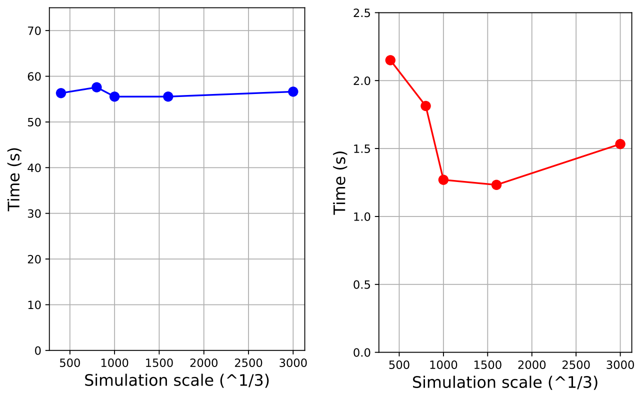
In-Situ Density PDF Calculation: We computed the Probability Density Function (PDF) in order to test in-situ analysis with heavy parallel computation. This calculation invokes parallel rasterization of all particles followed by a redistribution. The resulting output is a 2D image depicting the distribution function, rendered using Matplotlib [29] on the CPU.



(a) SZ, error bound = 1×10^{-3} (b) SZ, error bound = 1×10^{-5} (c) SZ, error bound = 1×10^{-7}

Fig. 6. Slice of a turbulence simulation with 3000^3 particles at t = 11.0 physical time. Data is compressed with SZ with different error bounds.

2D Particle Visualization: We visualized particles in a thin slice around $|Z| = 0$, comparable to the local spatial resolution where the colors represent the velocity magnitude $|v|$.



(a) (b)

Fig. 7. Time taken for in-situ generation of (a) probability density function, and (b) 2D slice for visualization in various simulation scales.

Results in Figure 7(a) show that the time to generate a PDF in-situ remains consistent as simulation scale increases, due to constant data processed per node during weak scaling. Figure 7(b) shows that the time to extract and visualize a 2D slice fluctuates slightly but remains roughly constant, indicating the architecture is efficiently scaling up with a highly balanced workload distribution.

Figure 8 shows that the lossy SZ compressor reduces 2D slice generation time as accuracy decreases. Despite data reduction, visualization time remains stable. Compression does not significantly speed up the process. Only at lower accuracy (e.g., 10^{-3}) the time for compression and visualization (5.99s) approach the time without compression (1.53s).

Figure 6 represents 2D visualizations generated using the aforementioned pipelines with 3000^3 simulation scale. As illustrated in Figure 6(a), when the error bound is high up to 10^{-3}, compression artifacts become clearly visible. Conversely, at 10^{-5} accuracy, the visualization retains almost the same level of detail as at 10^{-7} accuracy, bringing in an 81.2% reduction in transferred data size. The benefits of using lossy compressors to reduce data size during visualization are substantial for easier in-transit or post-hoc data analysis. However, for in-situ data analysis, the use of lossy compressors can introduce additional overhead due to the need for compressing and decompressing the data, requiring further analysis.

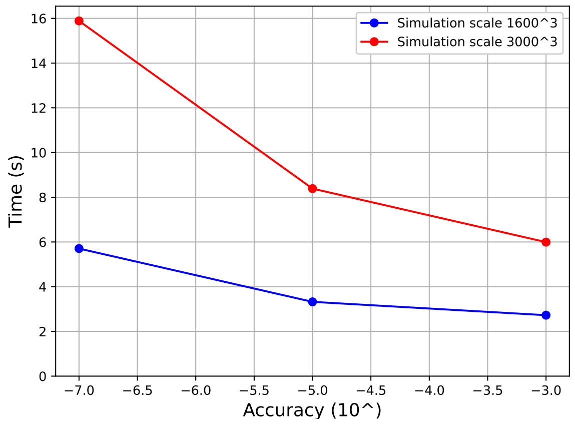


Fig. 8. Time for generating an in-situ visualization of a 2D slice, compressed with lossy SZ compressor, with different accuracy settings. 66 nodes were used for processing the 1600^3 simulation data. 432 nodes were used for processing 3000^3 simulation.

V. Conclusion

Compression and asynchronous I/O are two efficient solutions for reducing storage requirements and improving I/O performance in scientific simulations. However, most existing simulation frameworks do not integrate the two, limiting the further possibilities of extreme-scale in-situ data analysis. To address these challenges, we proposed an I/O architecture integrated into SPH-EXA, that enables parallel I/O and flexible data analysis linked by a compressed data flow. Future work will focus on extending compatibility to a broader range of computing architectures, and developing more domain-specific data analysis algorithms designed for SPH-based physical simulations.

Acknowledgment

This work is supported by the PASC (Swiss Platform for Advanced Scientific Computing) project ”SPH-EXA2: Optimizing Smoothed Particle Hydrodynamics for Exascale Computing” and the SKACH consortium (SERI). The numerical calculations were supported by the EuroHPC-JU Extreme Scale Access Mode through ”TGSF: The role of turbulence and gravity in star formation” (proposal ID: EHPC-EXT-2023E01-031). We also acknowledge support from Center for Scientific Computing in University of Basel (sciCORE), the LUMI Supercomputer Center and the Swiss National Supercomputing Center (CSCS - allocation c32).

References

[1] A. Cavelan, R. M. Cabezón, M. Grabarczyk, and F. M. Ciorba, “A smoothed particle hydrodynamics mini-app for exascale,” in *Proceedings of the Platform for Advanced Scientific Computing Conference*, 2020.

[2] “Piz daint — cscs,” www.cscs.ch, 05 2024. [Online]. Available: https://www.cscs.ch/computers/piz-daint/

[3] L. Wang, R. Xie, B. Chen, X. Yu, J. Ma, C. Li, Z. Hu, X. Sun, C. Xu, S. Dong *et al.*, “In-situ visualization of the space-charge-layer effect on interfacial lithium-ion transport in all-solid-state batteries,” *Nature Communications*, vol. 11, no. 1, p. 5889, 2020.

[4] J. Ravi, S. Byna, and M. Becchi, “Runway: In-transit data compression on heterogeneous hpc systems,” in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2023, pp. 229–239.

[5] X. Li, X. Meng, Z. Zhuang, G. He, L. Li, H. Jin, and L. Guo, “In situ visualization of salt crystallization in sub-/supercritical water environments,” *Desalination*, p. 117700, 2024.

[6] F. Meyer, B. Hernandez, R. Pausch, R. Widera, D. Groß, S. Bastrakov, A. Huebl, G. Juckeland, J. Kelling, M. Leinhauser *et al.*, “Hardware-agnostic interactive exascale in situ visualization of particle-in-cell simulations,” in *Proceedings of the Platform for Advanced Scientific Computing Conference*, 2023, pp. 1–14.

[7] M. Larsen, E. Brugger, H. Childs, and C. Harrison, “Ascent: A Flyweight In Situ Library for Exascale Simulations,” in *In Situ Visualization For Computational Science*. Cham, Switzerland: Mathematics and Visualization book series from Springer Publishing, May 2022.

[8] U. Ayachit, *The paraview guide: a parallel visualization application*. Kitware, Inc., 2015.

[9] “Catalyst documentation.” [Online]. Available: https://catalyst-in-situ.readthedocs.io/en/latest/introduction.html

[10] “Vtk documentation.” [Online]. Available: https://docs.vtk.org/en/latest/about.html

[11] J. Ahrens, K. Brislawn, K. Martin, B. Geveci, C. C. Law, and M. Papka, “Large-scale data visualization using parallel data streaming,” *IEEE Computer graphics and Applications*, vol. 21, no. 4, pp. 34–41, 2001.

[12] D. A. Huffman, “A method for the construction of minimum-redundancy codes,” *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[13] lz4, “lz4/lz4,” GitHub, 08 2019. [Online]. Available: https://github.com/lz4/lz4

[14] S. Di and F. Cappello, “Fast error-bounded lossy hpc data compression with sz,” in *2016 ieee international parallel and distributed processing symposium (ipdps)*. IEEE, 2016, pp. 730–739.

[15] P. Lindstrom, “Fixed-rate compressed floating-point arrays,” *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[16] X. Liang, B. Whitney, J. Chen, L. Wan, Q. Liu, D. Tao, J. Kress, D. Pugmire, M. Wolf, N. Podhorszki *et al.*, “Mgard+: Optimizing multilevel methods for error-bounded scientific data reduction,” *IEEE Transactions on Computers*, vol. 71, no. 7, pp. 1522–1536, 2021.

[17] P. Lindstrom, *ZFP: Fast, Accurate Data Compression for Modern Supercomputing Applications*. LLNL, 2023.

[18] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, “An overview of the hdf5 technology suite and its applications,” in *Proceedings of the EDBT/ICDT 2011 workshop on array databases*, 2011.

[19] W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck *et al.*, “Adios 2: The adaptable input output system. a framework for high-performance data management,” *SoftwareX*, vol. 12, p. 100561, 2020.

[20] P.-S. Yeh, W. Xia-Serafino, L. Miles, B. Kobler, and D. Menasce, “Implementation of ccsds lossless data compression in hdf,” in *Earth Science Technology Conference 2002*, 2002.

[21] S. Byna, M. S. Breitenfeld, B. Dong, Q. Koziol, E. Pourmal, D. Robinson, J. Soumagne, H. Tang, V. Vishwanath, and R. Warren, “Exahdf5: Delivering efficient parallel i/o on exascale computing systems,” *Journal of Computer Science and Technology*, vol. 35, pp. 145–160, 2020.

[22] H. Tang, Q. Koziol, J. Ravi, and S. Byna, “Transparent asynchronous parallel i/o using background threads,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 891–902, 2021.

[23] M. Dorier, Z. Wang, S. Ramesh, U. Ayachit, S. Snyder, R. Ross, and M. Parashar, “Towards elastic in situ analysis for high-performance computing simulations,” *Journal of Parallel and Distributed Computing*, vol. 177, pp. 106–116, 2023.

[24] Fides, “Fides: Adaptable data interfaces and services,” GitHub, 05 2019. [Online]. Available: https://gitlab.kitware.com/vtk/fides

[25] T. M. Inc., “Statistics and machine learning toolbox,” Natick, Massachusetts, United States, 2022. [Online]. Available: https://www.mathworks.com/help/stats/index.html

[26] H. Jasak, A. Jemcov, Z. Tukovic *et al.*, “Openfoam: A c++ library for complex physics simulations,” in *International workshop on coupled methods in numerical dynamics*, vol. 1000. Dubrovnik, Croatia), 2007, pp. 1–20.

[27] J.-l. Gailly and M. Adler, “Zlib compression library,” 2004.

[28] J. Tian, S. Di, K. Zhao, C. Rivera, M. H. Fulp, R. Underwood, S. Jin, X. Liang, J. Calhoun, D. Tao *et al.*, “Cusz: An efficient gpu-based error-bounded lossy compression framework for scientific data,” in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020, pp. 3–15.

[29] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in science & engineering*, vol. 9, no. 03, pp. 90–95, 2007.