

# Exascale In-Situ Visualization for Astronomy & Cosmology

Nicola Tuccari<sup>1,2,\*,†</sup>, Eva Sciacca<sup>1,†</sup>, Yolanda Becerra<sup>3,†</sup>, Enric Sosa Cintero<sup>3,†</sup> and Emiliano Tramontana<sup>2,†</sup>

<sup>2</sup>Università di Catania, Dipartimento di Matematica e Informatica, Catania, Italy

<sup>1</sup>INAF Astrophysical Observatory of Catania, Catania, Italy

<sup>3</sup>Barcelona Supercomputing Center, Barcelona, Spain

## Abstract

Modern simulations and observations in Astronomy & Cosmology (A&C) produce massively large data volumes, posing significant challenges for storage, access and data analysis. A long-standing bottleneck in high-performance computing, especially now in the exascale era, has been the requirement to write these large datasets to disks, which limits the performance.

A promising solution to this challenge is in-situ processing, where analysis and visualization are performed concurrently with the simulation itself, bypassing the storage of the simulation data.

In this work, we present new results from an approach for in-situ processing based on Hecuba, a framework that provides a highly distributed database for streaming A&C simulation data directly into the visualization pipeline to make possible on-line visualization. By integrating Hecuba with the high-performance cosmological simulator ChaNGa, we enable real-time, in-situ visualization of N-body simulation results using tools such as ParaView and VisIVO.

## Keywords

visualization, cosmology, in-situ visualization, high performance computing

## 1. Introduction

Astrophysical observations and simulation codes executed on high-performance supercomputers generate massive data volumes, often reaching petabyte scales. Managing such data poses considerable challenges that must be addressed to enable scientific discoveries[1]. Emerging pre-exascale infrastructures offer new possibilities for scaling applications in Astronomy and Cosmology (A&C), in particular to support high-performance visualization, which is essential for understanding simulation outcomes.

Traditionally, scientific visualization has relied on post-processing, where simulation data is first stored permanently on disk and only later retrieved for analysis and visualization. However, this type of workflow can be inefficient given the massive amount of data generated. An alternative processing paradigm is in-situ visualization, which enables the analysis and rendering of data in real time, concurrently to the execution of the simulation, thus allowing users to generate the visualization as the data is generated.

Processing such massive datasets is only feasible with the use of HPC resources. A major bottleneck in achieving high performance and scalability comes from the reliance on parallel file systems. Additionally, file-based workflows often introduce execution patterns that require strict synchronization and complex code, limiting the capability to adapt to new requirements or hardware changes.

A practical and emerging alternative for scientific computing are Key-Value (KV) databases, as they are particularly well suited for handling time-series and spatial datasets. Moreover, they allow to analyze partial results and react, for instance, by discarding a cosmological simulation as soon as a certain event occurs.

---

ITADATA-WS 2025: The 4<sup>th</sup> Italian Conference on Big Data and Data Science – Workshops, September 9–11, 2025, Turin, Italy

<sup>†</sup>These authors contributed equally.

✉ nicola.tuccari@inaf.it (N. Tuccari); eva.sciacca@inaf.it (E. Sciacca); yolanda.becerra@bsc.es (Y. Becerra); enric.sosacintero@bsc.es (E. S. Cintero); emiliano.tramontana@unict.it (E. Tramontana)

ORCID 0009-0004-7802-2602 (N. Tuccari); 0000-0002-5574-2787 (E. Sciacca); 0000-0003-2357-7796 (Y. Becerra); 0000-0002-7169-659X (E. Tramontana)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

By leveraging Hecuba, we enable a specific form of in-situ visualization known as in-transit visualization [2], where analysis and visualization are executed on dedicated I/O nodes. These I/O nodes receive the full simulation results, but they avoid to store them, but rather they generate new information from analysis or provide run-time visualization.

In previous work [3], we presented the first steps of the integration of the Hecuba platform within the ChaNGa high-performance cosmological simulator and the in-situ visualization of its N-body results with the ParaView and VisIVO tools. In this paper, we present further developments, focusing on the ParaView plugin and reducing VisIVO dependence on file-based data storing, to enable in-situ visualization.

## 2. Related Works

In-situ visualization has become increasingly important in the field of high-performance computing, as simulations now produce massive volumes of data that are impractical to store and analyze with just post-processing. Several surveys in the literature provide comprehensive overviews of existing techniques and tools for in-situ visualization, highlighting their central role in scientific workflows that demand real-time analysis while reducing I/O overheads [4].

Three main approaches to in-situ visualization can be identified: tightly coupled, loosely coupled and hybrid. In the tightly coupled model, visualization routines are directly embedded within the simulation code, which allows efficient memory access and minimal latency but requires synchronous execution and additional memory resources. The loosely coupled model, by contrast, separates computation and visualization into independent processes that may run on different resources and communicate through shared memory or network interfaces. Finally, hybrid approaches attempt to balance the advantages of both by performing data reduction during the simulation and forwarding the processed results to external resources for further analysis.

Several frameworks have been developed to support these approaches. One example is VisIt Libsim [5], which implements the tightly coupled model by enabling simulation codes to function as a compute engine directly linked to VisIt. Another widely adopted framework is ParaView Catalyst [6], which provides flexible integration mechanisms within simulation codes and supports both tightly and loosely coupled configurations.

These solutions have demonstrated the feasibility and advantages of in-situ workflows, but they also highlight persistent challenges, such as the complexity of code instrumentation and the risk of resource contention between simulation and visualization tasks.

## 3. Data, Tools and Methodology

### 3.1. ChaNGa Cosmological Data

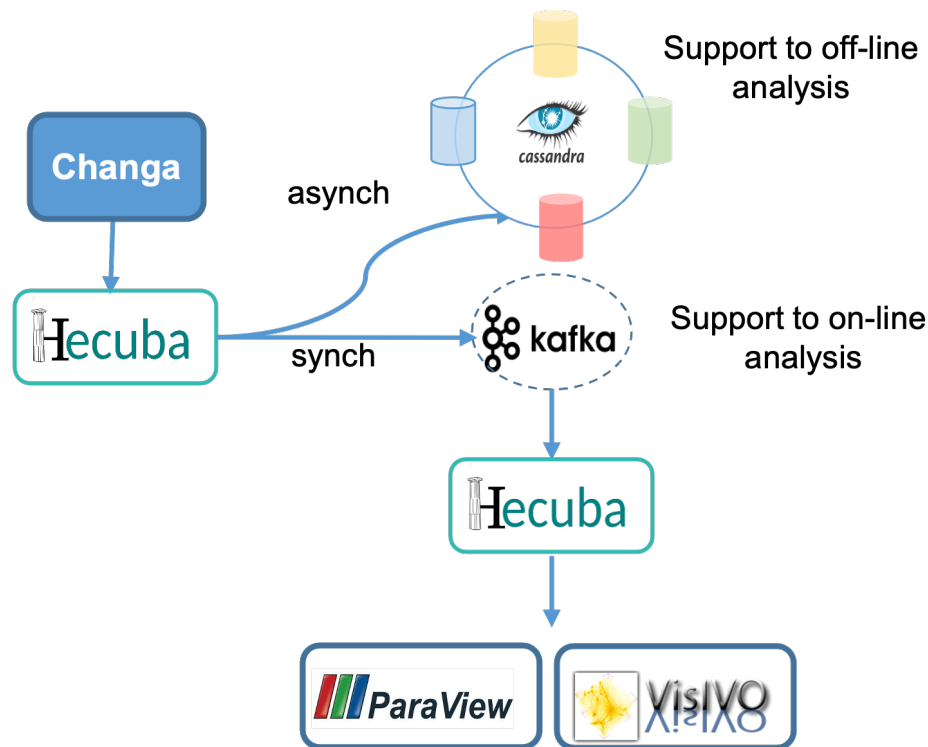
ChaNGa<sup>1</sup> (Charm N-body Gravity solver) is a parallel code designed for collisionless N-body simulations in astrophysics. It is capable of performing both cosmological simulations with periodic boundary conditions in comoving coordinates and simulating isolated stellar systems. It also includes hydrodynamics modeling using the Smooth Particle Hydrodynamics (SPH) method. Gravity calculations are computed using the Barnes-Hut tree algorithm, which includes hexadecapole expansion of nodes and Ewald summation for handling periodic forces. For time integration, ChaNGa uses a leapfrog integrator that allows each particle to evolve with its own individual timestep. ChaNGa stores data using Tipy, a binary file format used primarily in astrophysical simulations to store snapshot data of N-body systems, it can store three types of particles: gas, dark matter and star.

---

<sup>1</sup><https://github.com/N-BodyShop/changa>

### 3.2. Hecuba

Hecuba<sup>2</sup> is a set of tools and interfaces that aim to facilitate the management and utilization of persistent data for Big Data applications.



**Figure 1:** Hecuba architecture implementing an Object Mapper for Apache Cassandra support both off-line and on-line ChaNGa data analysis.

Hecuba implements an Object Mapper for Apache Cassandra [7] (a recognized NoSQL database) that enables programmers to interact with it and its data as regular in-memory objects. Currently, Hecuba implements an interface for Python and C++ programming languages. In order to provide a mechanism to synchronize run-time visualizations, we are extending Hecuba to implement a lambda architecture integrating Apache Kafka<sup>3</sup>. Lambda architecture is a data-processing structure defined to speed up online analysis for Big Data applications. With this approach, Hecuba allows at the same time to persist the generated data in a key-value datastore and also to produce a stream of data for online analysis. Notice that programmers can use the same interface to access the data, whether it is in memory, in storage, or it is coming through a stream: they only need to modify the class definition of the object to indicate which kind of object it is.

### 3.3. ParaView

ParaView<sup>4</sup>[8] is a tool designed to support the visualization and analysis of large scientific data sets. To this end, ParaView supports, for example, parallel data processing and rendering to enable interactive visualization or remote parallel computing. The decoupled architecture of ParaView allows it to run the setup as a client-server model with two separate processes: a server which runs on a potentially powerful remote machine and a ParaView client which runs on a desktop machine. Moreover, ParaView is an extensible framework that uses a complete plugin mechanism to allow the addition of new functionalities. In this work, we have implemented a custom ParaView plugin that uses the Hecuba interface to

<sup>2</sup><https://github.com/bsc-dd/hecuba>

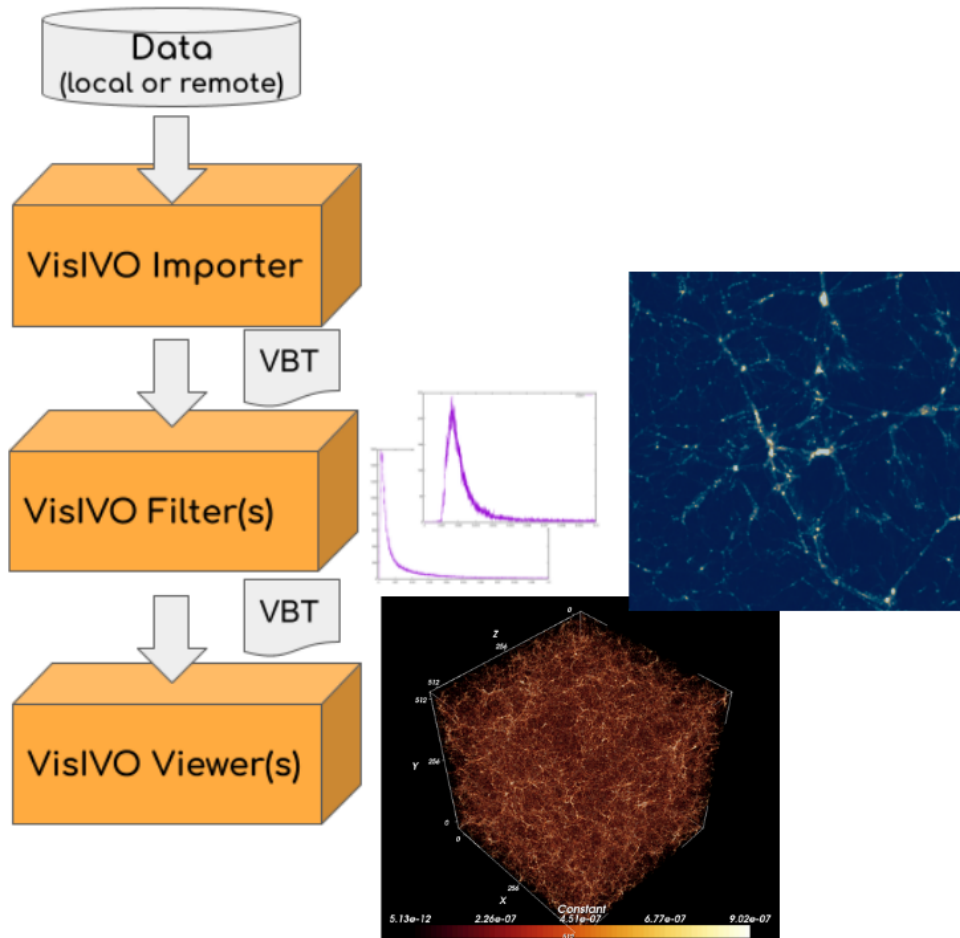
<sup>3</sup>Apache Kafka, <https://kafka.apache.org/>

<sup>4</sup><https://www.paraview.org/>

implement the online acquisition of the data to visualize. Also, to perform further comparisons for evaluation, we have extended this plugin to provide the option to read data from a Tipy file.

### 3.4. VisIVO

VisIVO is a software designed for high-performance, multi-dimensional visualization and analysis of large-scale astrophysical datasets[9]. VisIVO Server<sup>5</sup> stands out as a modular platform designed for creating customized views of 3D renderings from astrophysical datasets. It consists of three core components: VisIVO Importer, VisIVO Filter, and VisIVO Viewer. All of these components make use of the VisIVO Binary Table (VBT) data structure, a highly efficient data representation internally used by VisIVO Server. We are working on evolving VisIVO to take advantage of the capabilities provided by modern high-performance computing environments. A typical visualization pipeline performed with VisIVO Server modules is shown in figure 2.



**Figure 2:** Typical visualization pipeline of VisIVO Server consisting on the application of the three main modules.

We presented the integration of Hecuba within VisIVO Server in our previous work[10]. This integration allowed VisIVO Server to interact with a highly distributed database or use its streaming capabilities to perform in-situ visualization.

To complete the work to make VisIVO compatible with in-situ visualization, we needed to update VisIVO's implementation in order to avoid the reliance on the storage.

To reach this requirement, as a first step, we extended `VSTable`, a VisIVO Server internal data structure, to support in-memory data handling. The `VSTable` data structure is responsible for performing all

<sup>5</sup><https://github.com/VisIVOLab/VisIVOserver>

the required functions to modify or create a VBT. To support in-memory data handling, we developed a new subclass, `VSTableMem`, in which all columns are stored in memory using `std::vector`. Furthermore, all functions responsible for data access and manipulation, such as `PutColumn` and `GetColumn`, were redefined to operate on the in-memory data structures. This modification allows VisIVO to avoid relying on VBT files and, at the same time, it provides the flexibility to use both the file-based approach or the in-memory one by using subclassing.

The second step involved updating the VisIVO Library, a library designed to directly expose the VisIVO Server features within the user code. To take advantage of the new in-memory data handling capabilities, the API of the library was extended.

VisIVO Library is organized using environments, which are represented using a variable that contains all settings for a specific operation. At first, we needed to add a new environment setting to enable the module use the in-memory version of the importer and the viewer classes.

Furthermore, we implemented a new library function to allow users to retrieve in-memory vectors generated by the importer and pass them to the Viewer. The new function is defined as follows:

```
int VV_SetTableFromImporter(VisIVOViewer* viewer, VisIVOImporter* importer,
size_t tableIndex);
```

The first parameter is the `VisIVOViewer` environment, the second is the `VisIVOImporter` environment, and the third specifies the index of the desired table to retrieve. This is necessary because each `VisIVOImporter` can generate multiple tables, for example, if the simulation contains different particle types.

## 4. Preliminary Results

### 4.1. VisIVO Results

The previous preliminary results for VisIVO included the development of a prototype featuring an Importer using Hecuba's API to retrieve the simulated data from a distributed database.

The new results consist of a visualization pipeline built using the VisIVO Library, exploiting the newly implemented features to generate the rendering of the simulation data retrieved using the Hecuba importer without relying on storage.

The pipeline is composed of two steps, one that involves VisIVO Importer and another one involving VisIVO Viewer.

In the first step, the VisIVO Importer environment is defined and the required attributes are set. Specifically, we define the desired importer, in this case Hecuba, and the "use\_memory" attribute, which instructs the importer to use local memory. These are set using the following two library calls: `VI_SetAtt(&env, VI_SET_FFORMAT, "hecuba")` and `VI_SetAtt(&env, VI_SET_USE_MEMORY, "")`.

Next, the VisIVO Viewer environment is also defined with the required attributes, in this case the particle coordinates and the "use\_memory" attribute. As a final step, we use the new function that allows us to retrieve the data from the Importer,

```
VV_SetTableFromImporter(&viewerEnv, &importerEnv, 0).
```

In this case, it retrieves the first table that has 0 as index, which represents the available gas particles.

We compared the execution time of a pipeline using both the classic file-based approach and the in-memory approach (see Table 1). The pipeline consists of two steps: first, the ChaNGa importer processes two Topsy files containing 2.6 million particles (109 MB) and 161 million particles (6.38 GB); second, VisIVO's Points Viewer is executed on the gas particles. The file-based approach required 1.34 s and 39.11 s to complete the pipeline on the two datasets, while the in-memory implementation reduced this to 0.815 s and 29.76 s, respectively. In the file-based case, the importer step alone took 0.57 s and 9.86 s. Since the in-memory approach avoids one file write in the importer step and one subsequent read in the viewer step, these performance improvements are consistent with expectations.

# of particles	File-based	In-memory	Performance Gain
2.60E+06	1.34 s	0.815	39%
1.61E+08	39.11 s	29.76	24%

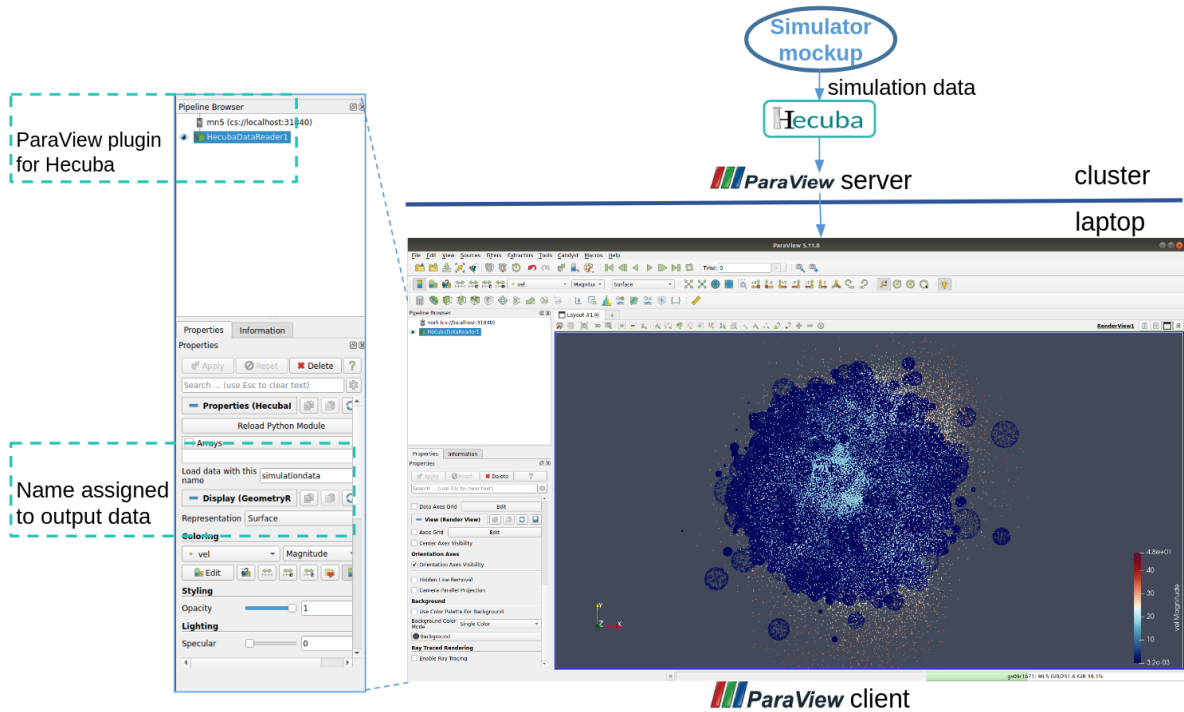
**Table 1**

VisIVO execution time comparison of a pipeline using both the classic file-based approach and the in-memory approach.

We expect to achieve similar results using the VisIVO’s Hecuba importer as soon as streaming capabilities are implemented on Hecuba’s C++ API.

## 4.2. ParaView plugin Results

Regarding ParaView, we have implemented a Python plugin in which the user is able to request the desired data to read from the stream, browse through all the different timesteps of the simulation and visualize them in the view window as shown in Figure 3. The goal is for the simulator to be able to



**Figure 3:** ParaView application window with the plugin with Hecuba integrated that supports in situ analysis.

run on the same infrastructure as the plugin or on a different one. Hecuba’s interface allows data to be visualized as it is received without user intervention. To test the plugin, we implemented a small code that acts as a data generator, as a producer. The generated data is the result of a previous ChaNGa simulation, stored in Topsy format files. Our producer reads these files and uses Hecuba’s interface to send them to our ParaView plugin via stream.

In addition, we have performed preliminary performance evaluations using two Topsy files, each containing 2.6 million particles (109MB) and 161 million particles (6.38GB). Our setup offers better performance than if ParaView had to read and manipulate the Topsy files (see Table 2). Under these conditions, the times required by the plugin to read from the stream the data from these files using Hecuba are 0.91s and 38.78s, respectively, while the times necessary to read directly from the Topsy file are 3.02s and 188.3s.

As part of our next steps, we plan to do a more exhaustive evaluation using more input data sizes.



# of particles	File-based	Hecuba streaming	Performance Gain
2.60E+06	3.02 s	0.91 s	70%
1.61E+08	188.3 s	38.78 s	79%

**Table 2**

Paraview execution time comparison of a pipeline using both the classic file-based approach and the Hecuba streaming approach.

We also aim to implement the interaction of ChaNGa with Hecuba so that it becomes the real producer in our experiments.

## 5. Conclusions and future work

In-situ processing is increasingly adopted to overcome the I/O bottlenecks associated with storing large-scale simulation outputs. This paradigm is especially useful in the context of Astronomy and Cosmology, where the capability to analyze data at runtime enables more efficient workflows.

This paper presented the development and refinement of an in-situ processing strategy leveraging the Hecuba toolset, integrated into a ParaView plugin and the VisIVO visualization environment, to enable run-time in-situ visualization simultaneously with data generation.

For future work in the context of VisIVO, we are working on adding the capability of in-memory data handling also to the VisIVO Filter module. This enhancement will enable more complex workflows, such as the generation of volumetric data using the data received by the Hecuba importer and then performing volume rendering without intermediate storage.

For the ParaView plugin, we are still working to optimize and refactor the code to have a better software pattern and maintenance. The small code that acts as a producer is a placeholder because we will study how to implement the interaction of ChaNGa with Hecuba to skip the usage of intermediate files. Also, in addition to the results shown, we aim to perform a more comprehensive evaluation using more input data sizes.

## Acknowledgments

This work is funded by the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain under grant agreement No 101093441 and it is supported by the spoke "FutureHPC & BigData" of the ICSC – Centro Nazionale di Ricerca in High Performance Computing, Big Data and Quantum Computing – and hosting entity, funded by European Union – NextGenerationEU.

## Declaration on Generative AI

During the preparation of this work, the author(s) used GPT-4 in order to: Grammar and spelling check. After using these tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

## References

- [1] T. Hey, S. Tansley, K. Tolle, J. Gray, The Fourth Paradigm: Data-Intensive Scientific Discovery, Microsoft Research, 2009. URL: <https://www.microsoft.com/en-us/research/publication/fourth-paradigm-data-intensive-scientific-discovery/>.
- [2] K. Moreland, R. Oldfield, P. Marion, S. Jourdain, N. Podhorszki, V. Vishwanath, N. Fabian, C. Docan, M. Parashar, M. Hereld, et al., Examples of in transit visualization, in: Proceedings of the 2nd international workshop on Petascale data analytics: challenges and opportunities, 2011, pp. 1–6.

- [3] N. Tuccari, E. Sciacca, F. Vitello, I. Colonnelli, Y. Becerra, E. S. Cintero, G. Marin, M. Jaros, L. Riha, P. Strakos, S. Trujillo-Gomez, E. Tramontana, R. Wissing, High performance visualization for astrophysics and cosmology, in: 2025 33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), 2025, pp. 443–450. doi:10.1109/PDP66500.2025.00069.
- [4] M. Rivi, L. Calori, G. Muscianisi, V. Slavnić, In-situ visualization: State-of-the-art and some use cases, 2012.
- [5] B. Whitlock, J. M. Favre, J. S. Meredith, Parallel in situ coupling of simulation with a fully featured visualization system, in: Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization, EGPGV '11, Eurographics Association, Goslar, DEU, 2011, p. 101–109.
- [6] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, J. Mauldin, Paraview catalyst: Enabling in situ data analysis and visualization, in: Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV2015, Association for Computing Machinery, New York, NY, USA, 2015, p. 25–29. URL: <https://doi.org/10.1145/2828612.2828624>. doi:10.1145/2828612.2828624.
- [7] A. Lakshman, P. Malik, Cassandra — A Decentralized Structured Storage System, Operating Systems Review. 44 (2010) 35–40.
- [8] J. P. Ahrens, B. Geveci, C. C. Law, Paraview: An end-user tool for large-data visualization., in: C. D. Hansen, C. R. Johnson (Eds.), The Visualization Handbook, Academic Press / Elsevier, 2005, pp. 717–731.
- [9] E. Sciacca, U. Becciani, A. Costa, F. Vitello, P. Massimino, M. Bandieramonte, M. Krokos, S. Riggi, C. Pistagna, G. Taffoni, An integrated visualization environment for the virtual observatory: Current status and future directions, Astronomy and Computing 11 (2015) 146–154.
- [10] N. Tuccari, E. Sciacca, Y. Becerra, E. Sosa Cintero, R. Wissing, S. Shen, E. Tramontana, In-situ high performance visualization for astronomy & cosmology, 2024. Presented at ADASS 2024.