Ref. Ares(2025)5204863 - 30/06/2025



Scalable Parallel Astrophysical Codes for Exascale

SPACE Visualisation data analysis and workflow framework #1 Release

Deliverable number: D3.2

Version 1.0/1.0





Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain under grant agreement No 101093441

Project Information

Project Acronym: SPACE
Project Full Title: Scalable Parallel Astrophysical Codes for Exascale
Call: Horizon-EuroHPC-JU-2021-COE-01
Grant Number: 101093441
Project URL: https://space-coe.eu

Document Information

Editor:	Guillermo Marín, BSC
Deliverable nature:	Report (R)
Dissemination level:	Public (PU)
Contractual Delivery Date:	30.06.2025
Actual Delivery Date	30.06.2025
Number of pages:	36
Keywords:	Data analytics, visualization, workflows
Authors:	Guillermo Marin – BSC Eva Sciacca – INAF Sebastian Trujillo Gomez – HITS Yolanda Becerra – BSC Enric Sosa Cintero – BSC Nicola Tuccari – INAF Petr Strakos – IT4I@VSB Milan Jaros – IT4I@VSB Miguel Esteban – BSC Iacopo Colonnelli – UNITO Doriana Medić – UNITO
Peer review:	Benoît Commerçon – CNRS Gino Perna – EngineSoft

History of Changes

Release	Date	Author, Organization	Description of changes
0.1	16.01.2025	Eva Sciacca, INAF	Provided template
0.2	19.02.2025	Guillermo Marin, BSC	Edited the summary
0.3	24.02.2025	Eva Sciacca, INAF	Drafted chapter 2.3
0.4	31.03.2025	Enric Sosa Cintero, BSC	Drafted chapter 2.1
0.5	29.04.2025	Milan Jaros & Peter Strakos, IT4I	Drafted chapter 2.2
0.6	12.05.2025	Doriana Medic, UNITO	Drafted chapter 3
0.7	31.05.2025	Doriana Medic, UNITO	Final chapter 3
0.8	02.06.2025	Miguel Esteban & Guillermo Marin, BSC	Final subsection 2.2.5
0.9	17.06.2025	all	Final edits
0.9.5	24.06.2025	Benoıt Commercon, CNRS; Gino Perna, ES	Internal review
1.0	30.06.2025	all	Final document



DISCLAIMER

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain. Neither the European Union nor the granting authority can be held responsible for them.

The space above and below the message intentionally is left blank.

Executive Summary

This deliverable, D3.2 "SPACE Visualisation data analysis and workflow framework #1 Release", is a report on the efforts in Task 3.1 and Task 3.3 towards developing the visualization and workflow tools proposed in the project proposal. It coincides with Milestone M7, which consists in the provisional but fully functional integration of post-processing and visualisation tools.

The development of our visualisation and workflow tools and solutions is guided by the work reported in D3.1 "SPACE ML, visualisation data analysis and workflow framework use cases and requirements", where we identified three use cases for visualisation and one for workflows, along with their respective user requirements. The main criteria for the current selection was: i) the relevance for the Astrophysics and Cosmology (A&C) community, and ii) that the use cases represent challenges in the field of visualisation, applicable to other scientific domains facing the same or similar challenges. A more thorough description of the use cases and requirements, as well as of the methods used to gather them, is available in D3.1.

In the following sections we discuss the process of developing this #1 release of the three Use Cases in visualisation and one in Workflows. For the sake of consistency, each use case has the following structure: Introduction, Tool development, Results, and Work plan for the next release. The use cases are discussed in the following order:

1. In-situ Parallel visualisation through Hecuba.
2. Interactive, volume-based cinematic visualisation.
3. Interactive, distributed visualisation for High Performance Computing (HPC) and cloud environments.
4. Topology-aware workflows for modular Exascale data analysis.

Contents

1 Introduction **6**

2 Visualisation Use Cases **7**
2.1 In-situ Parallel visualisation through Hecuba. 7
2.1.1 Introduction 7
2.1.2 Tool development 8
2.1.3 Results 12
2.1.4 Work plan for next release #2 13
2.2 Interactive, volume-based cinematic visualisation 13
2.2.1 Introduction 13
2.2.2 Tool development 14
2.2.3 Results 18
2.2.4 Work plan for next release #2 19
2.2.5 Additional exploration 19
2.3 Interactive, distributed visualisation for HPC and cloud 22
2.3.1 Introduction 22
2.3.2 Tool development 23
2.3.3 Results 25
2.3.4 Work plan for next release #2 29

3 Workflows Use Case **30**
3.1 Topology-aware workflows for modular Exascale data analysis 30
3.1.1 Introduction 30
3.1.2 Tool development 30
3.1.3 Results 31
3.1.4 Work plan for next release #2 33

4 Conclusions **33**

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

List of Figures

1 Hecuba setup with ParaView. . . . 8
2 Hecuba example: specification of a *StorageDict* and *StorageStream* in both python and C++. . . . 9
3 Hecuba example: producer in both python and C++. . . . 9
4 Hecuba data model. . . . 10
5 The same StorageDict in both python and C++. . . . 10
6 Producer code for the ChaNGa mockup. . . . 11
7 ParaView application window with the plugin. . . . 12
8 *smproxy.source* and subclassing *VTKPythonAlgorithmBase* does expose this code as a source module for ParaView. . . . 12
9 When we read from the stream using the *poll()* function provided by Hecuba. . . . 13
10 The BSpace add-on in Blender, where we selected a specific part of the data file (gas-velocity on the left and gas B-field on the right) and pressed the *Extract Data* button to obtain specific results from the SpaceConverter program. Left: Gas velocity visualisation from a ChaNGa cosmological simulation of a $(25\ \mathrm{Mpc})^3$ volume, containing 2.4×10^{10} particles. Based on Romulus25 initial conditions, run at higher resolution without subgrid physics. Right: Magnetic field (B-field) magnitude visualisation from a high-resolution isolated galaxy simulation using ChaNGa with full subgrid physics and MHD. Contains 1.68×10^8 particles, with $\Delta x = 5$pc resolution. . . . 14
11 The figure illustrates how the bounding box of BSpace add-on is used to select a specific region of the simulation. The visualized data represents gas with velocity fields, simulated using GADGET. The image on the right shows the selected area highlighted from the full dataset displayed on the left. . . . 15
12 Illustration of the SpaceConverter workflow in local mode, where data processing is performed on a remote machine, while rendering takes place locally on the user's computer within Blender. 17
13 Illustration of the SpaceConverter workflow in remote mode, where both data processing and rendering are performed on the remote machine. . . . 18
14 The figure shows the scalability performance of loading and converting a ChaNGa dataset into a volumetric format. . . . 19
15 The figure shows various visualisations of different particle types and properties from astrophysical simulations. From left to right: Dark Matter – Velocity, Gas – Masses, Gas – Velocity, Dark Matter – Masses, Disk – Masses, simulated using OpenGadget3. On the right is a galaxy with a magnetic field, simulated using ChaNGa. . . . 20
16 astroVDB conversion diagram . . . 20
17 Structural SIMilarity Index for the different output resolutions. Gas *velocity* field . . . 22
18 Renders of the sample simulation. Gas *velocities* field . . . 22
19 Typical visualisation pipeline in VisIVO (showing its three main modules) applied to OpenGadget3 cosmological simulation outputs. . . . 24
20 Preliminary scalability tests on 512GB Data. . . . 26
21 Visual representation of the VisIVO CWL workflows to generate particle renderings of ChaNGa output snapshots (on the left) and volume renderings of the OpenGadget3 output snapshots (on the right). The diagram lists all the inputs, outputs, and steps, and depicts the dependency relations between them. Workflows available at https://github.com/VisIVOLab/VisIVOCWL. . 27
22 Sample deployment of a VisIVO Workflow applied to OpenGadget3 cosmological simulation outputs deployed on HPC and Cloud infrastructures. . . . 28
23 Examples of VisIVO workflow outputs. Particles rendering of GAS particles from ChaNGa (left panel) and from OpenGadget3 (right panel). . . . 29
24 Examples of VisIVO workflow outputs of DEMNUni simulations. Volume rendering of CDM particles (left panel) and neutrino particles (right panel) at redshift $z = 0$. . . . 29
25 StreamFlow Workflow Management System . . . 31
26 The `streamflow.yml` file, template for deploying two steps on HPC and Cloud environment. . . . 32
27 xFFL: Cross-Facility Federated Learning. . . . 33

List of Tables

1 OpenVDB file sizes comparison . . . 21

2 Render times comparison. 21

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

List of Acronyms

A&C	Astrophysics and Cosmology
AI	Artificial Intelligence
BSC	Barcelona Supercomputing Center
CFD	Computational Fluid Dynamics
ChaNGa	Charm N-body GrAvity solver
CoE	Centre of Excellence
CWL	Common Workflow Language
GNNs	Graph Neural Networks
HPC	High Performance Computing
ICL	Intra Cluster Light
ML	Machine Learning
MN5	MareNostrum 5
SPACE	Scalable Parallel Astrophysical Codes for Exascale
SPH	Smooth Particle Hydrodynamics
VA	Visual Analytics
VBT	VisIVO Binary Table
VisIVO	Visualization Interface for the Virtual Observatory
WMS	Workflow Management System

1 Introduction

This deliverable D3.2 "SPACE Visualisation data analysis and workflow framework #1 Release" constitutes the initial report on the development and deployment of the visualisation data analysis and workflow framework envisioned within the Scalable Parallel Astrophysical Codes for Exascale (SPACE) Centre of Excellence (CoE), and provides an overview of the advancements made in creating scalable and efficient visualisation tools and workflows specifically designed for the A&C community.

The visualisation tools reported here, which are developed under SPACE Task 3.3 "High performance visualisation for astrophysics", aim to address critical challenges related to in-situ, cinematic, and distributed visual analytics to support the interpretation of predictions obtained from cosmological simulations, particularly within the context of Exascale computing. The workflow framework reported here, which is developed under SPACE Task 3.1 "Topology-aware workflows for modular HPC applications", supports the creation and execution of modular high-performance data analysis applications tailored for the next generation of Exascale computing systems, and focuses on enabling topology-aware co-scheduling of loosely coupled components within complex scientific applications.

The foundational basis for this release is the comprehensive identification of use cases and the corresponding user requirements gathered during the first year of SPACE, documented in D3.1 "SPACE ML, Visualisation and workflow use cases and requirements". D3.1 outlined crucial application scenarios including in-situ parallel visualisation, interactive volume based cinematic visualisation, and distributed HPC visualisation. The current Deliverable D3.2 presents significant progress made in these key areas corresponding to the three identified use cases in visualisation, and the one identified in Workflows:

- In-situ Parallel visualisation through Hecuba.
- Interactive, volume-based cinematic visualisation.
- Interactive, distributed visualisation for HPC and cloud environments.
- Topology-aware workflows for modular Exascale data analysis

This document details the methodologies, initial results, and evaluations for each use case, emphasizing modularity, scalability and adherence to FAIR principles for scientific data and codes (i.e. Findability, Accessibility, Interoperability, and Reusability). Furthermore, it outlines the planned enhancements and additional functionalities to be incorporated in the second and final release toward the end of the project. This deliverable serves, not anly as a milestone marking the successful first-stage deployment of these visualisation and workflow solutions, but also as a stepping stone towards new tools for the A&C community, tailored for research based on Exascale A&C simulations.

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

2 Visualisation Use Cases

2.1 In-situ Parallel visualisation through Hecuba.

2.1.1 Introduction

Traditional workflows prescribe storing the simulation results to disk and later retrieving them for analysis and visualisation. However, at exascale this storage of the full results becomes prohibitive. A solution to this problem is to run the analysis and visualisation concurrently (in-situ) with the simulation and bypass the storage of the full results. One mechanism for doing so is in-transit visualisation, where analysis and visualisation run on I/O nodes that receive the full simulation results but write information from analysis or provide run-time visualisation. The goal is to enable online visualisation of the data output of the simulator. This online analysis allows to check the partial results in advance so scientists do not need to wait until the simulation finishes to start analyzing the data.

In this project, Barcelona Supercomputing Center (BSC)[1] uses Hecuba[2][3] to adapt both the simulator and the visualisation tool to enable online visualisation. Hecuba is a set of tools and interfaces that implement a simple and efficient access to data stores for big data applications. Hecuba implements an Object Mapper for Cassandra[4], a recognized noSQL distributed database that allows programmers to use a common interface to access data as regular in-memory objects, regardless if they are persistent (stored in disk) or they are actual in-memory data.

In order to provide a mechanism to synchronize run-time visualisations, we have extended Hecuba to implement a lambda architecture integrating Apache Kafka[5]. A lambda architecture is a data-processing architecture defined to speed up online analysis for Big Data applications[6], without losing the ability to persist the output data of the applications. With this approach, Hecuba allows at the same time to persist the generated data in a key-value datastore and to produce a stream of data for online visualisation. Thus, Hecuba is able to support both offline and online data management, as seen in Figure 1.

The simulation code selected for this use case is Charm N-body GrAvity solver (ChaNGa)[7]. ChaNGa is a code to perform collisionless N-body simulations and hydrodynamics using Smooth Particle Hydrodynamics (SPH). It can perform cosmological simulations with periodic boundary conditions in comoving coordinates or simulations of isolated stellar systems. The development of this framework requires the implementation of a specific writer in ChaNGa that outputs the data files using Hecuba in a format also readable by the end visualisation software selected: ParaView[8].

ParaView is an open-source, multi-platform scientific data analysis and visualisation tool that enables analysis and visualisation of extremely large datasets. ParaView has two main features that we are fully exploiting:

- It is an end-user application with a distributed architecture that can be easily integrated with the user's desktop or other remote and parallel computing resources. This is achieved following a client-server model with two separate processes: a server which runs on a potentially powerful remote working machine, and a ParaView client which runs on your desktop machine. Both processes communicate with each other so that we can visualize from our computer the data generated in the working machine.

- It is an extensible framework that uses a complete plugin mechanism to allow for the addition of new functionalities. In this project, BSC is developing a new ParaView plugin that uses Hecuba to get the data to visualize. The main complexity of this plugin is to use the new features of Hecuba to perform online data analysis translating its data structures to ParaView's for visualisation and end up with a setup as in Figure 1. The goal is to make this transparent to the user and to provide scientists with a simple-to-use interface to visualize the simulation data as it is generated.

[1] BSC, https://www.bsc.es/

[2] P. Santamaria, L. Oden, E. Gil, Y. Becerra, R. Sirvent, P. Glock, and J.Torres. Evaluating the Benefits of Key-Value Databases for Scientific Applications. In Computational Science – ICCS 2019: 19th International Conference, Faro, Portugal, June 12–14, 2019, Proceedings, Part I. Springer-Verlag, Berlin, Heidelberg, 412–426

[3] Hecuba code, https://github.com/bsc-dd/hecuba

[4] A. Lakshman and P. Malik. Cassandra — A Decentralized Structured Storage System. Operating Systems Review. 44. Pp 35-40. 2010

[5] Apache Kafka, https://kafka.apache.org/

[6] Lambda Architecture, https://softwarepatternslexicon.com/kafka/17/1/6/1/

[7] ChaNGa, https://faculty.washington.edu/trq/hpcc/tools/changa.html

[8] Ahrens, James, Geveci, Berk, Law, Charles, ParaView: An End-User Tool for Large Data Visualisation, Visualisation Handbook, Elsevier, 2005, ISBN-13: 9780123875822

All these elements ultimately form the setup shown in Figure 1. First, we have the simulator, in our use case ChaNGa, generating its output. Then, thanks to Hecuba utilizing the Cassandra and Kafka APIs, this output is written into the database for support in offline analysis and also into the stream for support in online analysis. Finally, thanks to a custom ParaView plugin with Hecuba integrated, we read from the stream and start visualizing the results. The database and Kafka server are launched by a Hecuba tool (explained later in Section 2.1.2) in the running cluster(s) and run independently of the use of Hecuba and thay are shared by both the simulator and ParaView.

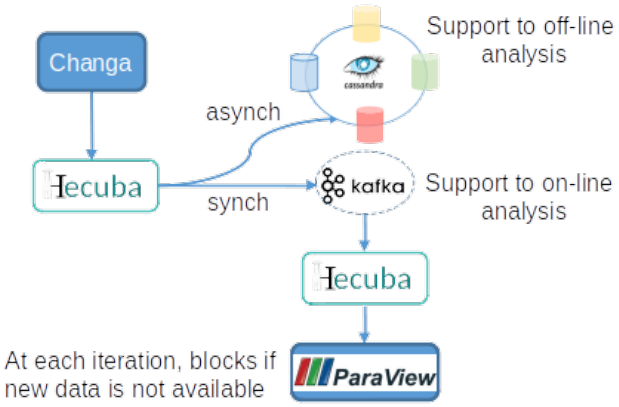


Figure 1: Hecuba setup with ParaView.

2.1.2 Tool development

In this section, we show the data models and methods used with the components previously mentioned: Hecuba, ChaNGa and its mockup, and ParaView and its plugin. In addition, we touch upon our setup and working machine. The following subsections describe more extensively each of these tools.

Hecuba As described above, Hecuba enables programmers to interact with Cassandra and its data as regular in-memory objects in Python and C++. Hecuba implements some basic classes that hide the underlying storage system and implements the data organization in the storage system: *StorageDict* (equivalent to Python and C++ dictionaries), *StorageNumpy* (equivalent to Numpy Ndarrays[9] in Python) and *StorageObject* (equivalent to regular in-memory objects). Any class derived from one of these basic classes becomes a Hecuba backed class. Thus, a program using Hecuba can be composed of the following steps:

1. Define (if required) a user-defined class derived from one of the basic Hecuba classes.
2. Instantiate an Hecuba object. The programmer can choose if the instance is volatile or persistent.
3. At any moment a volatile Hecuba object can become persistent using the Hecuba method *make_persistent*. The parameter of *make_persistent* is the name that will identify the object in the data store.
4. Access the Hecuba object as a regular Python object, using its underlying Python interface (for example, the selector operator ([]) for the *StorageDicts* or the Numpy functions to operate with *StorageNumpys*).

In this work, we also use Hecuba's streaming functionality. To do so, Hecuba integrates into its code Apache Kafka, an open-source distributed event streaming platform. Currently, we have this feature in work in progress for *StorageDict* and *StorageNumpy*. The user has to define the class as a subclass of both *StorageDict* and *StorageStream* Hecuba classes and make the corresponding object persistent. With this class definition, all writings will send the data to the storage asynchronously and through the stream synchronously. Figure 2 shows a simple example that uses both *StorageDict* and *StorageStream* in Python and C++.

In Kafka's terminology, there are the producer and the consumer. To put it simply, the producer is the one who writes into the stream and the consumer is the one who reads from it.

[9]NumPy, https://numpy.org/

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

```
class mydict (StorageDict, StorageStream):
  '''
  @TypeSpec dict <<key:int>,val:numpy.ndarray>
  '''
```

(a) Hecuba example: declaration of a *StorageDict* and *StorageStream* in python.

```
using Key = KeyClass<int>;
using Value = ValueClass<StorageNumpy>;
class mydict: public StorageDict <Key,Value,Dict>, public StorageStream {};
```

(b) Hecuba example: declaration of a *StorageDict* and *StorageStream* in C++.

Figure 2: Hecuba example: specification of a *StorageDict* and *StorageStream* in both python and C++.

In Hecuba, the user, as the consumer, should use the function *poll* to receive the streamed data. What *poll* does is wait until it can read something from the stream. At the current state, the consumer is only supported in python.

On the other hand, the producer is supported in both python and C++ as shown in Figure 3. The code for the producer is the same than when the streaming capability is not set.

```
d=mydict("dictname")
d[42]=sn #sn is a StorageNumpy
```

(a) Producer in python.

```
mydict d;
d.make_persistent("dictname");

// setting of k and sn

d[k] = sn;
```

(b) Producer in C++.

Figure 3: Hecuba example: producer in both python and C++.

The data model chosen for the data management of the simulation uses the *StorageNumpy* to contain the data of all the particles (this model will be explained later in this text). When sending Hecuba classes over the stream, Hecuba first sends the identifiers to do the instantiation at the destination and then the data values. This implies that for a *StorageNumpy* the whole numpy ndarray is sent, which can cause problems when these are very large.

One problem is that the Kafka stream is unable to transmit messages that are too large —numpy ndarrays too large in this case—. To solve this problem, we have modified the Hecuba producer and consumer code. In the producer code, before sending a message over the stream, it detects if the message is too large. If it is, it splits the data into smaller messages and then sends them to the stream. So, instead of trying to send a message with a numpy ndarray that is too large, we send several smaller ones. Finally, the consumer code has been modified to handle these cases when it needs to read a numpy ndarray from the stream. In this way, Hecuba can transmit heavier data per stream.

ChaNGa mockup ChaNGa's output consists out of particles positions, velocities and other properties per timestep. For each output timestep, a main binary output file is generated along with auxiliary binary or ascii (user can decide) files. The main binary output file follows the TIPSY[10] format, which is structured with a header containing some metadata and then the particle data: first gas particles, then dark matter particles, and then star particles with their respective properties for each.

As a placeholder, for this project we programmed a mockup that reads from these TIPSY files and writes the data they contain to the Hecuba data structures. The information for the code to read from the TIPSY files has been obtained from this repository[11]. Notice that this means that this code will act as a producer, writing in the stream the data read from the TIPSY.

[10] TIPSY, https://faculty.washington.edu/trq/hpcc//tools/tipsy/tipsy.html
[11] https://github.com/N-BodyShop/flatiron-tests

This program has been introduced as a placeholder because for now this solution allows us to test the plugin and its setup with ease. In the next steps, we are going to integrate Hecuba into ChaNGa so that its output is written directly to Hecuba, bypassing the file system involved in reading the TIPSY files.

An easy and intuitive way to store timesteps and their data is to use a *StorageDict*. Each dictionary entry would have the timestep as a key and the particle data as a value. However, depending on the simulation, there might be too many particles and data to hold in a single dictionary entry. To solve this, we divide the space occupied by the particles into subareas (similar to an octree approach) and have a dictionary entry for each subarea (Figure 4). Therefore, we would have a key —an entry identifier— composed of a timestep and its bounding box, defining its corresponding subarea. Notice that in Figure 4, *Xmin*, *Ymin*, *Zmin*, *Xmax*, *Ymax* and *Zmax* define the bounding box of a three-dimensional area. As a value, we again use one of Hecuba's classes: *StorageNumpy*, a two-dimensional numpy ndarray in which a particle would be a row and each particle attribute would be a column (for example, x coordinate, y coordinate, z coordinate, velocity information, density, etc.).

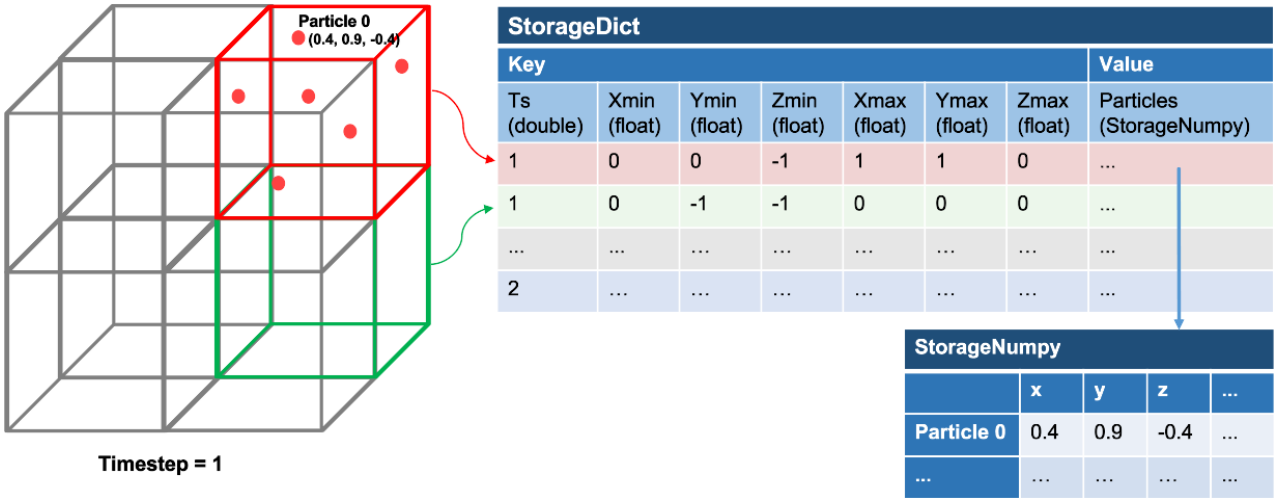


Figure 4: Hecuba data model.

Therefore, to hold the simulation data we would have the *StorageDict* as seen in Figure 5.

```
class Dict(StorageDict, StorageStream):
    '''
    @TypeSpec dict <<ts:double, x_min:float, y_min:float, z_min:float, x_max:float,
    y_max:float, z_max:float>, particles:hecuba.hnumpy.StorageNumpy>
    '''
```

(a) The StorageDict definition in python.

```
using Key = KeyClass<double, float, float, float, float, float, float>;
using Value = ValueClass<StorageNumpy>;
class Dict: public StorageDict <Key, Value, Dict>, public StorageStream {};
```

(b) The StorageDict definition in C++.

Figure 5: The same StorageDict in both python and C++.

Next, in Figure 6 we show the instantiation of the dictionary, its explicit persistence into the database, and the writings into it within this mockup. In addition, the construction of the *StorageNumpy* is shown that acts as the value of a *StorageDict* entry. Notice that the code for the producer is the same as when the streaming capability is not set, so even though the syntax is not explicit it still writes to the stream as long as the *StorageDict* has been instantiated with *StorageStream*[12].

[12] https://github.com/bsc-dd/hecuba/wiki/1:-User-Manual

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

```
Dict data; //Dict class defined before
data.make_persistent("mydict"); //so it persists the object and writes into stream

for (i = 0; i < NUM_AREA; ++i) {
  //np_data: it contains all the data of the particles
  //meta: metadata needed for the StorageNumpy
  StorageNumpy storageNumpy(np_data[i], meta);

  storageNumpy.make_persistent(snName); //so it stores into the database

  //prepare the key for the entry, composed of a timestep and a bounding box
  //areas: metadata needed for the Key
  Key k(areas[i].ts, areas[i].min.x,
        areas[i].min.y, areas[i].min.z,
        areas[i].max.x, areas[i].max.y,
        areas[i].max.z);
  Value v(storageNumpy); //prepare the value for the entry
  data[k] = v; //write the StorageNumpy into the StorageDict and stream
}
```

Figure 6: Producer code for the ChaNGa mockup.

ParaView plugin With its extensive plugin mechanism, ParaView allows us to incorporate our needs and tools into its structure. To do so, our plugin —transparent to the user— translates the Hecuba data structures explained before to VTK-format so ParaView can output the results. When the plugin is loaded, it will start Hecuba, that is to say, it will initialize the connection with the database and the stream, thus setting up off-line and on-line data analytics respectively as shown in Figure 1.

Within this plugin, as normally, the user uses the ParaView application window to display the visualisation of data. In this project, we have developed for this plugin that the user will be able to request the data he wants to read from the stream, browse through the different timesteps of the simulation and visualize them in the view window. The ChaNGa mockup mentioned above will have defined the *StorageDict* with a specific name. Then, through an extension in the ParaView UI thanks to the plugin, the user will be able to submit the same name of the dictionary and be able to read from it from the stream as can be seen at the bottom left of Figure 7. Once we have selected the corresponding data we will visualize the first timestep of the simulation. From this point on, we will be able to navigate through the timesteps of the simulation using the standard ParaView GUI buttons. Per timestep, all particles will be loaded in PointCloud format.

As for the implementation, the ParaView's user manual[13] recommends the use of Python for its ease of use and portability. For a convenient way to package our Python-based modules, and provide them with user interfaces for the future, then the *VTKPythonAlgorithmBase*-based approach is recommended. Here, you write a Python class subclassing *VTKPythonAlgorithmBase* and implementing methods to do data processing, just like any other VTK-based filter or source.

The module that will contain this plugin is a source module since it ingests data into the system from the stream. To expose this source in ParaView, you have to add decorators to the class definition as Figure 8 shows.

The class that inherits from *VTKPythonAlgorithmBase* needs several methods to function by the programmer, the most important being *RequestInformation* and *RequestData*. In the *RequestInformation*, metadata such as the number of timesteps in the dataset and their temporal values for the temporal datasets are managed. In particular, at this stage, the total number of timesteps for the simulation has been hardcoded to expedite the development of the plugin. Then we have the *RequestData* function, which does almost all the heavy lifting and be the algorithm expected to produce the output dataset as a PointCloud in our case. If a timestep has not yet been read, it will be requested from the stream using the *poll()* function as seen in Figure 9.

Cluster deployment The BSC disposes the services of MareNostrum 5 (MN5)[14], a supercomputer that can fulfill the requirements of any HPC user. This computational muscle will be necessary because the running framework handles data analysis and visualisation for large datasets. MN5 is a supercomputer with two main

[13] https://docs.paraview.org/en/latest/ReferenceManual/pythonProgrammableFilter.html
[14] MareNostrum 5, https://www.bsc.es/marenostrum/marenostrum-5

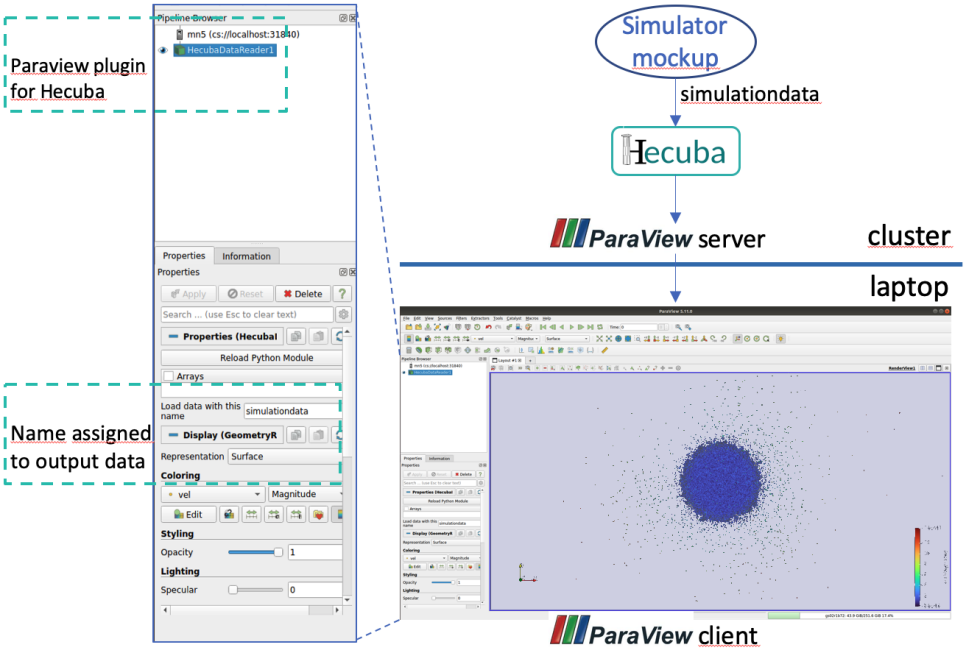


Figure 7: ParaView application window with the plugin.

```
@smproxy.source(name="HecubaDataReader", label="Hecuba - Data Reader")
class StorageDictReader(VTKPythonAlgorithmBase):
```

Figure 8: *smproxy.source* and subclassing *VTKPythonAlgorithmBase* does expose this code as a source module for ParaView.

partitions: the General Purpose Partition (GPP), our framework, and the Accelerated Partition (ACC). The system is supplied by Bull SAS combining Bull Sequana XH3000 and Lenovo ThinkSystem architectures.

There are two different types of filesystem available from a node: 1) a distributed filesystem (GPFS) accessible from all nodes, and 2) a local filesystem used for temporary storage during jobs execution.

The versions of each component of the software stack are the following: Red Hat Enterprise Linux 9.2 (Plow), Apache Cassandra 4.0-beta2, Hecuba 2.1.3, Apache Kakfa 2.8.0 and ParaView 5.11.0.

Before deploying ParaView on MareNostrum5, we have to set the cluster we are going to work with. To do that, Hecuba also has a tool called *c4s* (short for Cassandra4Slurm), that provides automatic deployment of a Cassandra cluster in a slurm-based system. This includes setting up the Kafka server, alongside Apache Zookeeper[15] ensemble to make it work.

Earlier in this document, it was stated that we were going to use ParaView's feature of being able to deploy our framework to this cluster. The basic concept is that if a dataset does not fit on a desktop machine due to memory or other limitations, we can split the dataset among a cluster of machines, managed from our desktop. To do this, we launch the client-server mode mentioned above. Specifically, we use an *ssh* connection to connect from our local computer to the *pvserver* launched in the cluster setup by *c4s*.

2.1.3 Results

The implementation of streaming capability in Hecuba has been completed. Thanks to this new functionality, Hecuba allows the transmission of data by stream to support on-line analysis, while also persisting the data in a database to support off-line analysis as usual. With this, we enable visualisation to run at the same time as data generation.

Moreover, the implementation of the ChaNGa mockup allows us to read any TIPSY file and write its data to the stream with ease. This approach has helped us to develop and to test the ParaView plugin with Hecuba

[15] https://kafka.apache.org/documentation

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

```
def RequestData(self, request, inInfo, outInfo):
  [...]
  numpys = []
  for i in range(self._boxes_per_ts): #number of subareas per timestep
    ts, xmin, ymin, zmin, xmax, ymax, zmax, v = self._storagedict.poll()
    numpys.append(v)
    [...]
  np.concatenate(numpys, axis=0) #returns a numpy
```

Figure 9: When we read from the stream using the *poll()* function provided by Hecuba.

integrated, which allows us to receive data from the stream and visualize it.

In addition, we have performed a preliminary performance evaluation using a TIPSY file of 2.6 million particles (which occupies 109MB). To perform this evaluation, we have extended our ParaView plugin to provide the option to read data from a file (as well as receive it from a stream). Measurements were taken using a single MareNostrum5 GPP node (General Purpose Partition). MareNostrum5 GPP consists of 6408 compute nodes, each one equipped with:

- 2x Intel Sapphire Rapids 8480+ at 2GHz and 56c each (112 cores per node)
- 256 GB of Main memory, using DDR5 (with 216 nodes with 1024GB)
- 1x NDR200 interconnection shared by 2 nodes (SharedIO) (BW per node 100Gb/s)
- 960GB on NVMe storage (/gpfs/scratch)

Note that the TIPSY input file was stored in the GPFS file system.

We measured the time required to read the data and to convert it to ParaView structures, both using the function of our ParaView plugin that reads from the stream using Hecuba and the one that reads from TIPSY files. Our setup offers better performance than if ParaView had to read and manipulate the TIPSY file. In these conditions the time used by the plugin to read from the stream using Hecuba is 0.91s while the time necessary to read directly from the TIPSY file is 3.02s.

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

2.1.4 Work plan for next release #2

For the ParaView plugin, we will optimize and refactor the code to have a better software pattern and maintenance.

Moreover, earlier in Section 2.1.2 we mentioned that the Changa mockup is a placeholder because we will study how to implement the interaction of ChaNGa with Hecuba to skip the usage of intermediate files. In this way, we would bypass the file system involved in reading the TIPSY files. The goal would be to translate the ChaNGa output directly to the corresponding Hecuba classes, writing directly to the stream.

Also, in addition of the results shown in Section 2.1.3, we want to do a more comprehensive evaluation using more input data sizes.

2.2 Interactive, volume-based cinematic visualisation

2.2.1 Introduction

Astrophysical and cosmological simulations require significant computational power and typically produce extremely large datasets, often ranging from terabytes to even petabytes. To derive meaningful scientific insights from such simulations, efficient post-processing and visualisation workflows are essential. Among the available post-processing techniques, volume rendering has emerged as a powerful method for visualizing complex physical phenomena. It enables researchers to visualize the entire simulation domain at once, making it possible to interpret trillions of calculated values through an intuitive and informative visual format. Volume rendering not only enhances understanding of spatial relationships in the data but also supports interactive manipulation and analysis, which is especially valuable in the study of large-scale cosmic structures and fluid dynamics.

To address the challenges associated with visualizing massive simulation outputs, we are developing a scalable high-performance visualisation workflow. This approach focuses on converting data from large-scale astrophysical and cosmological simulations into volumetric formats suitable for advanced volume rendering. The goal is to enable interactive visualisation using a reduced path-tracing setup (fewer samples per pixel, lower number of volume scattering bounces, etc.) that still delivers high visual fidelity, allowing users to adjust shader parameters, lighting, and camera perspectives in real time. Once a suitable visualisation configuration is achieved, a full-resolution offline path-tracing render can be used to produce final high-quality outputs.

Our visualisation workflow is built around Blender [1] and its Cycles [2] rendering engine. By leveraging Blender's powerful, open-source rendering capabilities and integrating it with high-performance computing (HPC) resources, we provide a hybrid system in which computationally intensive processing is carried out on remote clusters while the user interacts with the data through Blender on a local machine. This model not only reduces the demands on local hardware resources, but also allows users to maintain full control over the visualisation process.

To make this workflow possible, we have extended Blender with a custom add-on that facilitates the loading, configuration, and rendering of volumetric data generated from astrophysical and cosmological simulations. Blender's cross-platform support and compatibility with a wide range of hardware, including GPUs from NVIDIA, AMD, and Intel, ensures broad accessibility and performance consistency. Through this approach, we aim to provide researchers with an intuitive and efficient visualisation tool that bridges the gap between large-scale simulation data and high-quality cinematic rendering, enabling clearer insights and more engaging scientific communication.

2.2.2 Tool development

In this section, we introduce the tools that are currently being developed specifically for this project: the BSpace add-on, the CyclesPhi renderer with the BSpace extension, and SpaceConverter.

BSpace This add-on[16] (see Fig.10) provides a user-friendly interface within Blender for working with simulation data processed by the SpaceConverter. It enables users to load, configure, and render volumetric datasets such as OpenVDB [3] files directly inside Blender. OpenVDB is an open-source library developed by DreamWorks Animation [4] for efficient storage and manipulation of sparse volumetric data, commonly used in visual effects and computer graphics. Blender natively supports the OpenVDB format by default, allowing users to import, render, and manipulate sparse volumetric data. Figure 10 showcases two simulation use cases. On the left, we display a cosmological box with gas velocity visualisation. On the right, we show a high-resolution, non-cosmological isolated galaxy with gas and magnetic field (B-field) visualisation.

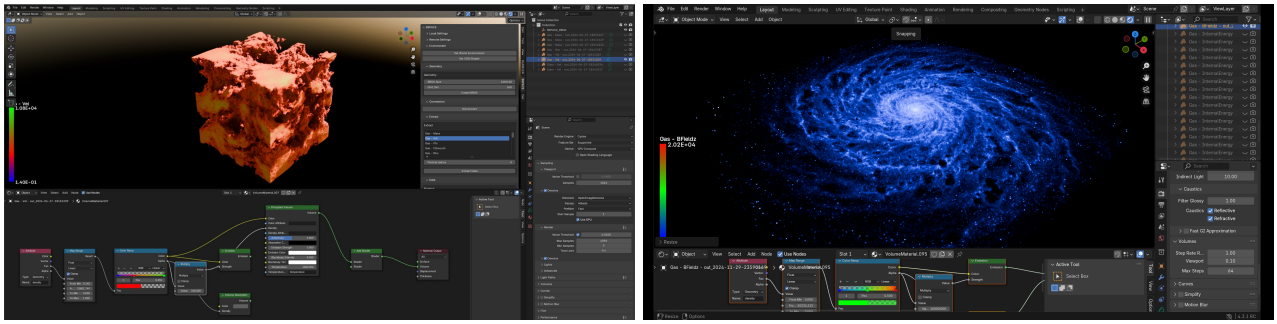


Figure 10: The BSpace add-on in Blender, where we selected a specific part of the data file (gas-velocity on the left and gas B-field on the right) and pressed the *Extract Data* button to obtain specific results from the SpaceConverter program. Left: Gas velocity visualisation from a ChaNGa cosmological simulation of a $(25\ \mathrm{Mpc})^3$ volume, containing 2.4×10^{10} particles. Based on Romulus25 initial conditions, run at higher resolution without subgrid physics. Right: Magnetic field (B-field) magnitude visualisation from a high-resolution isolated galaxy simulation using ChaNGa with full subgrid physics and MHD. Contains 1.68×10^8 particles, with $\Delta x = 5\mathrm{pc}$ resolution.

The interface is divided into several menu sections. The first section, *Local Settings*, allows the user to

[16] https://code.it4i.cz/blender/space_converter/-/wikis/usage-guide/bspace

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

specify the path to a local temporary folder where cached and intermediate files will be stored. This path can be entered manually or selected using a file browser.

The second section, *Remote Settings*, is used to connect Blender to a running SpaceConverter server for remote or cluster-based data conversion. The user can input the IP address or hostname of the server, with the default set to localhost for local usage. The communication port for the server can also be specified.

In the *Environment* section, users can automatically configure Blender's world settings to optimize volume rendering. This includes adjusting the background and lighting settings.

The *Geometry* section allows users to define and create a 3D bounding box (see Fig. 11) for the volumetric data. The bounding box size can be set in Blender scene units. The resolution of the volume grid can also be specified; for example, a value of 100 would create a grid of $100 \times 100 \times 100$ voxels. After entering the desired values, users can generate the bounding box, which defines the region of interest and selects the corresponding data for visualisation (see Fig. 11).

The *Connection* section manages communication with the server. Users can establish a connection to either a remote or local SpaceConverter server in order to load and process data. This connection can also be terminated when no longer needed.

The *Extract* section is used to select and extract volumetric data from the connected server. After establishing a connection, a list box displays the available data types (e.g. Gas - Velocity). Users can select a method for interpreting density. The particle radius can be adjusted for SPH representations, and a density multiplier can be set. Once the settings are chosen, the user can start the extraction process.

In the *Data* section, users can apply and manage the extracted data within the Blender scene. Material properties are applied to the volumetric object by default. Users can select an existing bounding box in the scene and reposition it to the current location of the 3D cursor.

The final section, *Select VDB*, displays a list of loaded OpenVDB files that are available for use in the scene. Each item in the list represents a separate volume dataset. Users can select one of these files to assign it to the bounding box or use it for rendering and inspection.

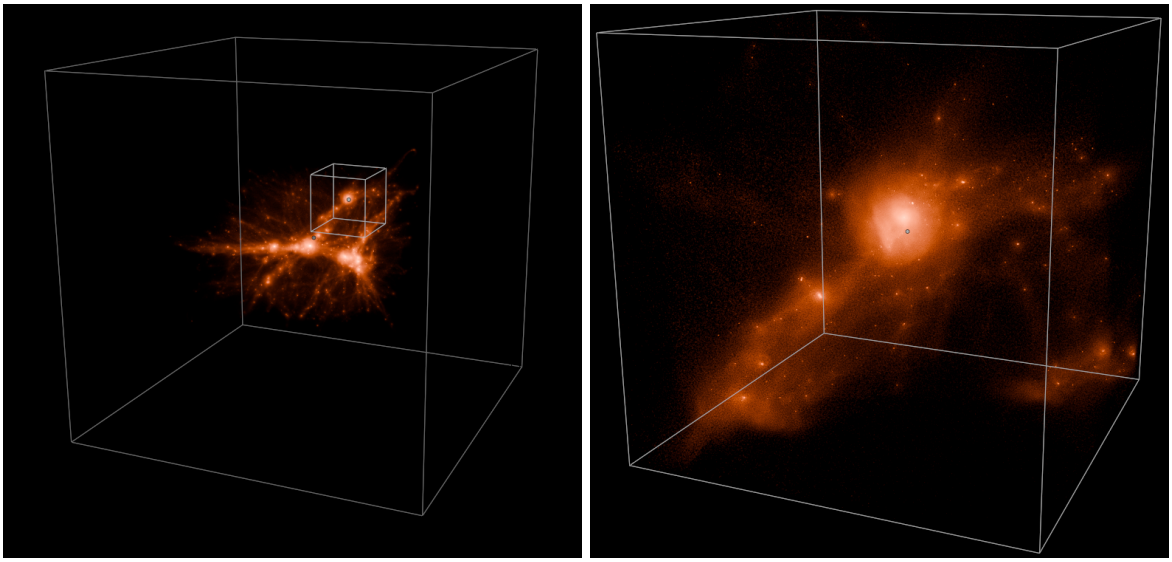


Figure 11: The figure illustrates how the bounding box of BSpace add-on is used to select a specific region of the simulation. The visualized data represents gas with velocity fields, simulated using GADGET. The image on the right shows the selected area highlighted from the full dataset displayed on the left.

CyclesPhi We have modified the core of the Blender Cycles rendering engine to extend its functionality for high-performance computing (HPC) environments. This improved version, called CyclesPhi[17], was developed at IT4Innovations. CyclesPhi has been optimized for efficient use of HPC resources through the integration of the following technologies: OpenMP, MPI, and CUDA Unified Memory. As a modified version of the original Blender rendering core, CyclesPhi is tightly integrated with Blender and is capable of rendering Blender scenes

[17] https://code.it4i.cz/raas/cyclesphi

by distributing the rendering load across multiple GPUs on multiple nodes using MPI. Each GPU is handled by one OpenMP thread.

As part of this project, we extended CyclesPhi with the BSpace extension, which serves as a bridge between the BSpace add-on, the SpaceConverter application and the CyclesPhi renderer. More information about the BSpace extension is provided below.

SpaceConverter SpaceConverter[18] is a software application developed for processing simulation data in the fields of astronomy and cosmology. It provides users with a range of tools to efficiently convert and prepare large-scale simulation outputs into formats suitable for visualisation and rendering.

A key feature of SpaceConverter is its support for simulation data formats such as OpenGadget3[19] and ChaNGa. It includes specialized importers that allow seamless handling of these cosmological formats. When working with OpenGadget3 data, users can provide a parameter file and snapshot file, and manage memory usage by specifying limits for maximum memory, buffer size, and particle allocation. For ChaNGa simulations, the application supports both the TIPSY format, requiring the path to the input file, and the NChilada [5] format, which requires the path to the corresponding data directory.

The application is compatible with multiple operating systems, including Linux, macOS, and Windows. It is primarily operated via a command-line interface, which enables users to convert simulation data into volumetric formats for further analysis and visualisation. Users can customize the output resolution by setting grid dimensions and specifying the output location as a file or directory. Additionally, remote operations are supported by configuring a server address and communication port. The application can also display general information about the simulation data being processed. Users may choose to convert the data into NanoVDB [6] format as an alternative to the default OpenVDB format. NanoVDB is a lightweight, GPU-optimized version of the OpenVDB library, designed for efficient real-time processing and rendering of sparse volumetric data. It maintains compatibility with OpenVDB structures while enabling high-performance applications on modern graphics hardware. Blender supports NanoVDB through the Cycles renderer, allowing for efficient GPU-based rendering of volumetric effects using NanoVDB grids. Dense data can be written directly to files, and specific datasets and particle types can be exported with options for either sparse or dense data representation. For greater efficiency, a simplified density calculation method is available.

Another important capability of SpaceConverter is its ability to perform high-performance conversions of Smoothed Particle Hydrodynamics (SPH) simulation data into volumetric VDB files, enabling more advanced and detailed visualisation techniques. The application is also fully integrated with Blender, providing compatibility with the Cycles rendering engine. This allows users to work with complex volumetric datasets directly within Blender's interface. SpaceConverter is designed for high-performance computing (HPC) environments and supports MPI-based parallelization, which allows the software to scale effectively across multiple CPU nodes. The application is capable of quickly producing detailed, high-resolution volumetric datasets that are immediately ready for rendering.

SpaceConverter includes two modes of operation, offline mode, where users can specify export parameters directly via the command line, and interactive mode, where parameters are set through a graphical interface integrated into Blender.

The interactive mode includes two sub-modes: Local Mode and Remote Mode. These determine where the rendering takes place, either on the user's local machine, or on a remote computing cluster. The main advantage of the remote mode is that the data stays on the remote cluster or computer and is not transferred to the user's local machine. This makes it possible to work with larger astrophysical datasets that might not fit on a personal device. Additionally, during the entire workflow, the data is not written to storage (neither locally nor remotely) which helps to accelerate the visualisation process.

Local Mode The workflow of the local mode (see Fig.12) begins on an HPC (High-Performance Computing) Cluster, where raw simulation data, typically in the OpenGadget3 or ChaNGa particle formats, is first loaded across multiple compute nodes. This stage is referred to as the *Loading (particles)* step, where each node handles a portion of the input data in parallel to optimize for large-scale datasets.

Once the particle data is loaded, it proceeds to the *Voxelization* stage. In this step, the particle data is converted into volumetric grid representations (also known as voxels) on each compute node. The voxelized output

[18] https://code.it4i.cz/blender/space_converter
[19] OpenGadget3: https://www.space-coe.eu/codes/opengadget.php

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

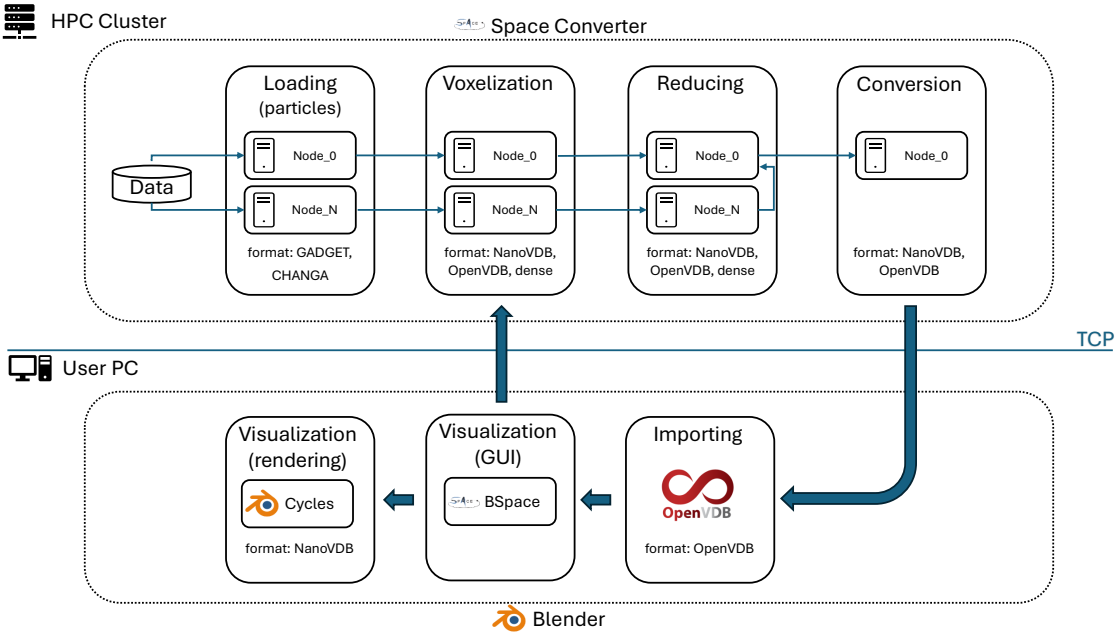


Figure 12: Illustration of the SpaceConverter workflow in local mode, where data processing is performed on a remote machine, while rendering takes place locally on the user's computer within Blender.

can be produced in several formats, including NanoVDB, OpenVDB, and dense grid structures, depending on the intended downstream usage.

Following voxelization, the data enters the *Reducing* stage, where it is aggregated or consolidated across nodes. This reduction step ensures that the voxel data from different nodes is merged efficiently, resulting in a unified volume dataset that remains in either NanoVDB, OpenVDB, or dense volumetric grid.

The next step is *Conversion*, which takes place on a single (root) node. Here, the reduced volume data is transformed into a final format suitable for transfer and visualisation, specifically NanoVDB or OpenVDB. After this conversion, the data is sent via TCP to the user's local machine for further use.

For interactive exploration, the user can utilize the BSpace GUI, which provides a graphical interface for viewing and manipulating volumetric data. For high-quality visual output, the user can initiate *visualisation (Rendering)* using Cycles, the rendering engine integrated with Blender. This rendering step specifically uses the NanoVDB format for optimized performance and quality.

Throughout the process, the system ensures efficient handling of large-scale volumetric simulation data, from initial loading and voxelization on an HPC cluster, to final visualisation and rendering on a user's local machine.

Remote Mode The workflow of the remote mode (see Fig.13) begins in HPC Cluster 1 (tested on Karolina CPU partition), where raw simulation data in OpenGadget3 or ChaNGa particle format is loaded in parallel across multiple nodes during the *Loading (particles)* stage. This parallelism (MPI) allows for efficient handling of large-scale datasets.

Once loaded, the data proceeds to the *Voxelization* stage, where each node converts the particle-based data into a volumetric format. The output of this stage is either in NanoVDB or a dense volume format. The voxel data is then passed to the *Reducing* stage, where it is aggregated across nodes to produce a more compact, unified representation. The reduced output also remains in either NanoVDB or dense format.

After reduction, the data enters the *Conversion* stage, where it is transformed into the NanoVDB format for portability and rendering. This converted data is then transmitted via TCP data stream to the next stage of the pipeline.

On the user's PC, the workflow includes several components that support GUI interaction, viewport visualisation, and both real-time and offline rendering for effective data visualisation and interaction. The GUI component, using BSpace, allows the user to interactively explore and navigate the data. At the same time, a

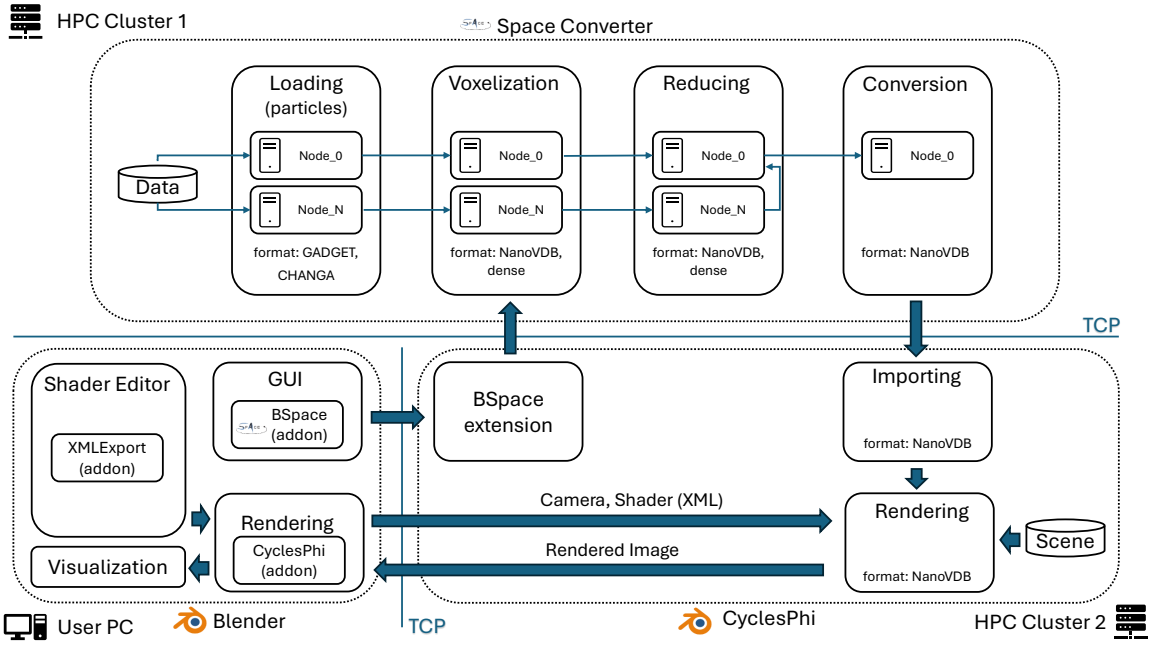


Figure 13: Illustration of the SpaceConverter workflow in remote mode, where both data processing and rendering are performed on the remote machine.

Shader Editor (enhanced by an XMLMaterial[20] add-on) lets users design and export rendering shaders in XML format.

The *BSpace extension*[21] of CyclesPhi acts as a bridge between the GUI and rendering systems, sending both the volume data (from the HPC cluster) and the user-defined shader (XML) to a remote rendering service over TCP. This rendering service is handled on HPC Cluster 2 (tested on Karolina GPU partition), which is optimized for rendering tasks. The *Importing* module receives the NanoVDB data and prepares it for the rendering stage. In the *Rendering* stage, the NanoVDB volume is processed along with the shader (XML) definitions, and the resulting rendered image is sent back to the User PC.

On the user's side, CyclesPhi, the customized Blender rendering add-on, is used to handle and display the rendered image, completing the workflow with final *visualisation* of the volumetric data.

2.2.3 Results

In this section, we show how effectively SpaceConverter works on a cluster and what results can be achieved in visualisation.

In Figure 14 we present the scalability results for data loading and basic conversion to a voxel grid over multiple MPI processes. The tests were carried out on the Karolina cluster at IT4Innovations. At peak performance, the configuration involved 64 CPU nodes, each node running 128 MPI processes, totaling 8192 MPI processes. The results show that the algorithm exhibits near-linear scalability. A slight performance drop at higher MPI counts is attributed to I/O limitations, as many processes attempt to access a single file simultaneously on a SCRATCH file system. This network issue can be mitigated by distributing the data across multiple I/O resources, e.g. more CPU nodes.

In Figure 15 we present a series of visualisations depicting different particle types and physical properties from large-scale astrophysical and cosmological simulations. These renderings demonstrate the effective transformation of particle-based data into volume-based cinematic visualisations, highlighting both scientific detail and visual impact. From left to right, the first five images represent various components of a simulation carried out using the OpenGadget3 code. These include dark matter velocity fields, gas mass distribution, gas velocity fields, dark matter mass distribution, and the distribution of stellar disk masses. Each visualisation highlights specific aspects of the simulation data, making it possible to identify structures such as filaments, halos, and

[20] https://code.it4i.cz/raas/cyclesphi/-/wikis/usage-guide/xmlmaterial_addon
[21] https://code.it4i.cz/raas/cyclesphi/-/wikis/usage-guide/space_extension

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

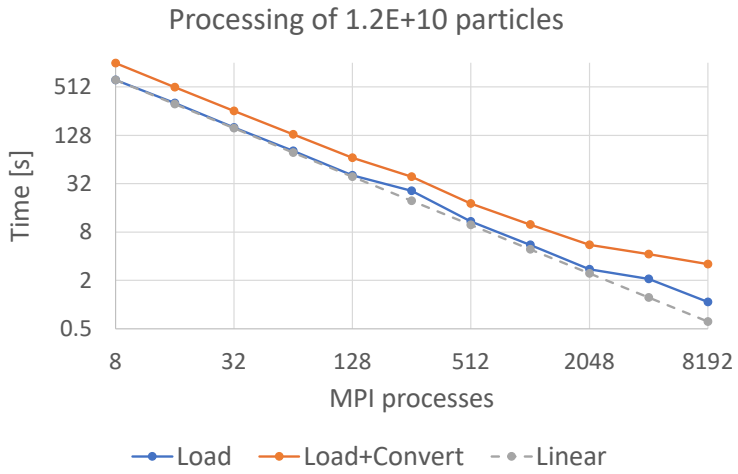


Figure 14: The figure shows the scalability performance of loading and converting a ChaNGa dataset into a volumetric format.

galactic formations. The application of color and lighting techniques in these renderings enhances the perception of density, motion, and spatial relationships within the simulated universe. The final image on the right shows a highly detailed galaxy with a magnetic field structure, generated using the ChaNGa simulation code. Together, these images demonstrate the versatility and visual richness achievable through the use of volume rendering and advanced data conversion tools within a visualisation workflow.

The results achieved within this project were presented at [7], [8], [9], [10].

2.2.4 Work plan for next release #2

In future work, the development of SpaceConverter and its Blender integration through the BSpace add-on will continue with several planned enhancements aimed at expanding its usability, performance, and visualisation capabilities. One of the goals is to extend support for additional simulation data generic formats such as HDF5, and VTK. These additions will significantly broaden the range of astrophysical and cosmological datasets that can be visualized and analyzed within our pipeline. Another planned improvement involves the extension of dense data processing to support additional kernel functions beyond the currently implemented models. This will allow for more accurate and flexible density representations, especially for complex Smoothed Particle Hydrodynamics (SPH) simulations. Further optimization of the data pipeline is also a priority, with a focus on reducing memory footprint and increasing processing speed during both conversion and rendering stages. In addition, support for animation workflows will be introduced, enabling users to visualize time-evolving simulations directly within Blender, including seamless playback and frame-by-frame rendering. These improvements will make SpaceConverter and BSpace even more powerful tools for researchers working with large-scale astrophysical data, ensuring a more efficient, flexible, and visually rich workflow for both scientific and creative applications.

2.2.5 Additional exploration

Data conversion. In addition to the previous pipeline, we have conducted an additional exploration of the OpenVDB format applied to the creation of cinematic visualisations of scientific data, which exploits a point-based tree structure known as OpenVDB Points. The resulting pipeline, named AstroVDB, is a software application designed to facilitate the conversion of scientific Smooth-Particle Hydrodynamics (SPH) simulation datasets, such as those generated by OpenGadget3 and ChaNGa, directly into particle-based formats. Its primary aim is to allow scientists to convert specific regions or entire simulations, reducing the total particle count to a predetermined value while preserving the relative proportions of distinct particle types, thus enabling the visualisation of large scale simulations on consumer-grade devices with minimal or no degradation in visual quality. OpenVDB Points is a configuration of OpenVDB that stores spatially organized arrays of points using a special OpenVDB tree, offering improved access speed and data compression compared to traditional linear

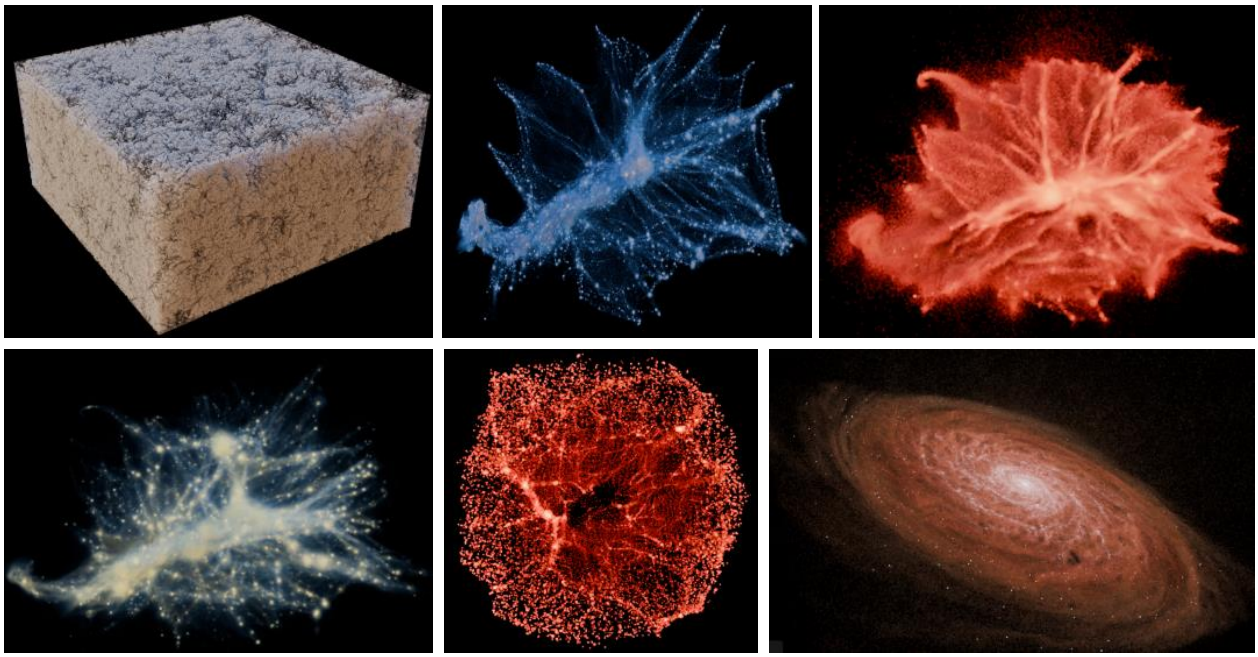


Figure 15: The figure shows various visualisations of different particle types and properties from astrophysical simulations. From left to right: Dark Matter – Velocity, Gas – Masses, Gas – Velocity, Dark Matter – Masses, Disk – Masses, simulated using OpenGadget3. On the right is a galaxy with a magnetic field, simulated using ChaNGa.

point arrays. This particle-based approach may offer several potential advantages: (a) its ability to preserve the original data representation, accelerating the conversion process, (b) faster render times compared to volumes, and (c) it maintains comparable visual fidelity to volumetric representations when down-sampling large datasets. Users can customize the conversion process via a human-readable `YAML` configuration file, specifying parameters such as the data source, target particle count, count tolerance, particle types, fields to extract, and regions of interest. Format-specific settings are also available for each supported simulation type, including OpenGadget3 and ChaNGa, with custom readers tailored to each format. OpenGadget3 simulations require the user to set the snapshot file path, memory limit, buffer size, and particle rate, whereas ChaNGa simulations —Tipsy and NChilada file formats— only require the data directory path. Configuration samples for all supported formats are available in the project's Git repository[22].

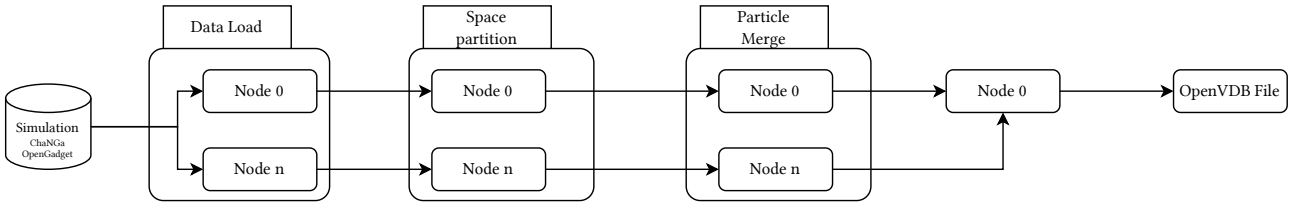


Figure 16: astroVDB conversion diagram

The tool is intended to run in the same clusters used for the simulations as a post-process, retrieving the resulting OpenVDB files after the conversion is finished to visualize them. During the conversion process, the simulation data is loaded in parallel across all nodes using MPI —as illustrated in the *Data load* stage of figure 16. Once loaded, all local particles for each type of object (gas, stars, dark matter, etc.) are spatially divided into voxels using an iterative, parallel, deterministic radix-based partitioner —see *Space partition* in figure 16— which ensures consistent results for identical inputs, and allows for controlling the total number of particles produced based on the user-defined final count. When reducing the number of particles, each type of object is

[22] https://gitlab.bsc.es/dataviz/severo-ochoa-film-productions/astrovdb

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

reduced proportionally to the original one, maintaining the distribution of the simulation data. In this process, a new reference particle is created on each voxel and its position is averaged between all particles inside the voxel. To calculate the field values of this reference particle, we use the radius of the voxel as the support domain, the particles inside the voxel as its integration domain, and an SPH kernel function produces the new interpolated field values of the reference particle. Finally, all reference particles are gathered to the root rank for sequential writing to OpenVDB files, with a separate file per frame and simulation-defined particle type, containing all selected fields for that type.

visualisation. Our current implementation is tested in SideFX Houdini[23], a visual effects and animation software widely used in the entertainment industry, as it offers first-class support for the OpenVDB format and is able to load both OpenVDB grids and OpenVDB Points. It also offers a high-end level of control over the image production, with features common to Blender. Within Houdini, the user has an array of different tools for visual manipulation, such as transfer functions to color, transparency or particle radius, based on the data properties. Houdini also offers additional advanced tools for clipping and slicing, or shading and illumination.

Preliminary results. Preliminary tests on visual fidelity and rendering performance show promising results. File size and image fidelity tests were performed with a ChaNGa-TIPSY simulation with a disk footprint of 46.29 GB that contains 1.612×10^8 particles. The size for each particle configuration is calculated as the total size of all produced files. All available fields have been converted for each particle type to have a fair comparison against the original simulation data.

Particles	Original	1M	5M	10M	20M	30M	50M	100M	161M
File size (MB)	46290	39.1	191.8	382.5	767.5	1146.2	1880.5	3829.0	5769.5
Reduction	-	1183.89	241.35	121.01	60.31	40.38	24.49	12.08	8.02

Table 1: OpenVDB file sizes comparison

Table 1 shows a 8x decrease in disk footprint against the original simulation data for the conversion only —161 Million particles— and a maximum reduction of 1184x for the smallest generated version —1 Million particles. Render tests were performed on an Apple M2 Pro (10 CPU cores) using Houdini 20.5 and an image resolution of 1280×720 pixels. Table 2 shows the rendering times and memory required by the renderer, CPU Karma 20.5 to produce the images. As expected, rendering times and memory footprint decrease with the reduction of particles.

Total particles	1M	5M	10M	20M	30M	50M	100M	161M
Load memory (GB)	0.658	0.768	0.974	1.33	1.45	2.2	3.56	4.15
Render time (s)	19	27	27	28	28	29	30	31

Table 2: Render times comparison.

To assess the visual similarity between rendered outputs of simulations conducted at varying particle resolutions, we employ the Structural SIMilarity Index (*SSIM*) [11]. Unlike traditional pixel-wise metrics, SSIM evaluates images based on perceived changes in luminance, contrast, and structural information, making it well-suited for comparing images that may differ in local information but are expected to exhibit similar global features. In our case —where files with differing numbers of particles are used to approximate the same physical phenomena— SSIM provides a perceptually meaningful metric to quantify structural consistency, allowing us to evaluate how lower-resolution versions capture the visual characteristics of higher-resolution ones. Figure 17 shows the SSIM index between the original images and the lower resolution versions. For global overviews of the simulation, a version with 6% of the original data —10M particle version, 2.53M gas particles— produces no perceptible loss in visual fidelity compared to the original, and for close up renders of smaller regions, it occurs with 12% of the original data —20M particle version, 5.06M gas particles.

Figure 18 contains samples of the produced imagery for the visual quality analysis with different particle counts for a general view and a close-up of a region of the simulation.

Work plan for next release #2. Future development will focus on extensive performance and scalability tests with a broader range of simulation datasets, including varying sizes and configurations from both

[23] https://www.sidefx.com/docs/

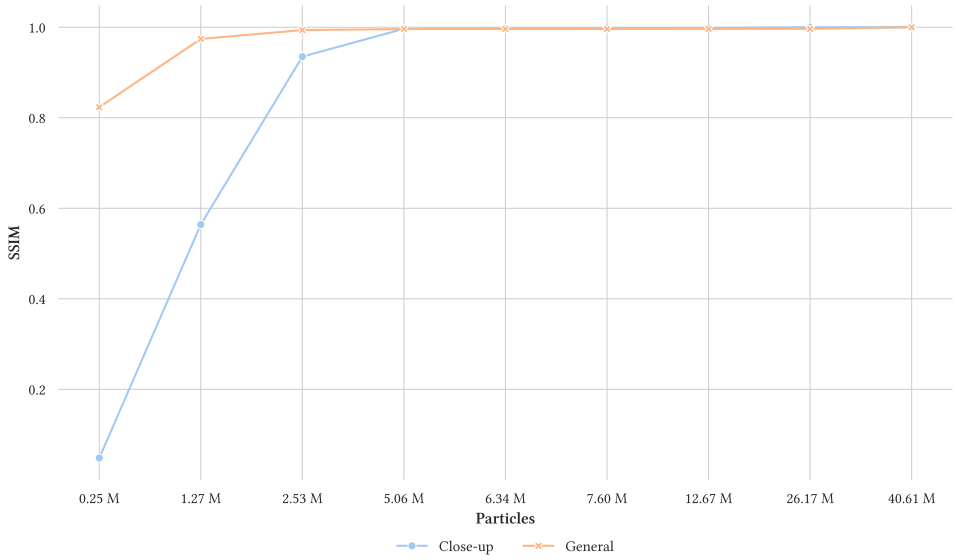
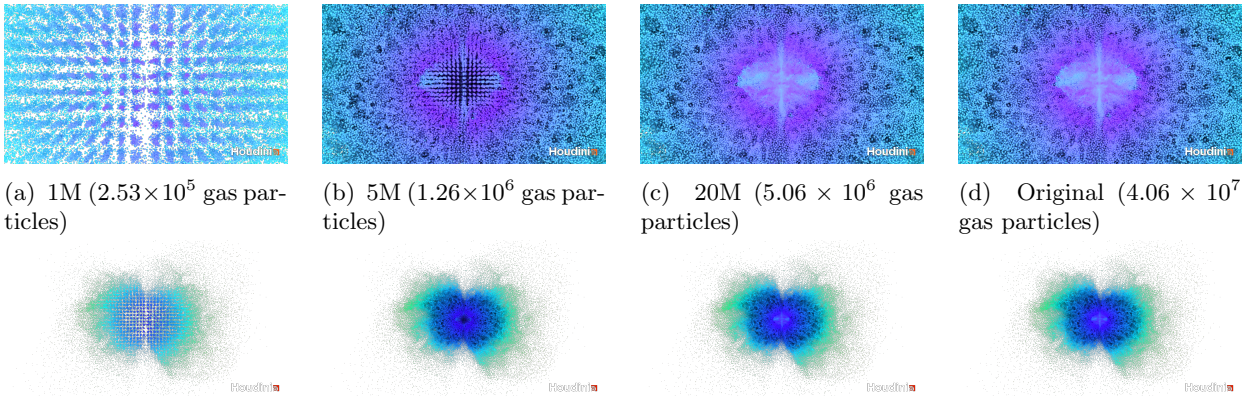


Figure 17: Structural SIMilarity Index for the different output resolutions. Gas *velocity* field



(a) 1M (2.53×10^5 gas particles)

(b) 5M (1.26×10^6 gas particles)

(c) 20M (5.06×10^6 gas particles)

(d) Original (4.06×10^7 gas particles)

(e) 1M (2.53×10^5 gas particles)

(f) 5M (1.26×10^6 gas particles)

(g) 20M (5.06×10^6 gas particles)

(h) Original (4.06×10^7 gas particles)

Figure 18: Renders of the sample simulation. Gas *velocities* field
First row contains General view, second row closeup region.

OpenGadget3 and ChaNGa, to ensure robustness and consistency across different scenarios and analyze the performance of astroVDB. One limitation of the current approach is that OpenVDB Points are not currently supported in many visualisation softwares. In this regard, additional output formats are under consideration to improve compatibility with other scientific visualisation tools. We also plan to explore integration with interactive visualisation platforms to support real-time rendering and collaborative analysis.

2.3 Interactive, distributed visualisation for HPC and cloud

2.3.1 Introduction

SPACE codes are expected to produce massively large data volumes (in the order of petabytes) executed on high performance computers. Such data volumes pose significant challenges for storage, access and data analysis. The visual exploration of big datasets, as one aspect of data analysis, pose some critical challenges as well, specifically: (i) Interactivity. The ability to deal with datasets exceeding the local machine's memory capacity; for complex visualisations the relevant computations should be performed close to the data to avoid time consuming streaming of large data volumes; (ii) Integration with the scientists' toolkit for seamless usage, abstracting from technical details related to the underlying HPC resources and freeing scientists to concentrate

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

in scientific research; (iii) Facilitate visualisation, processing and data analysis in a collaborative manner within, e.g., science gateway technologies to allow collaborative activity between users and provide customization and scalability of data analysis/processing workflows, hiding underlying technicalities.

INAF Astrophysical Observatory of Catania has been developing and maintaining the Visualization Interface for the Virtual Observatory (VisIVO)[24] since 2005 and it has been recently been extended with the ViaLactea Visual Analytic modules [12, 13]. VisIVO is developed adopting the Virtual Observatory standards[25] and its main objective is to perform 3D and multi-dimensional data analysis and knowledge discovery of a priori unknown relationships between multivariate and complex astrophysical datasets.

The use case evolution and the related implementation activities aligned with the following functional requirements:

- Enhance the portability of the VisIVO modular applications and their resource requirements. VisIVO modules, i.e., importer, filter(s) and viewer, are being parallelized to further exploit HPC and Exascale infrastructures. Improving its portability and potential of integration with other astrophysical pipelines and computing resources will make VisIVO fully integrated within the scientists' toolkit for its seamless usage, abstracting from technical details and freeing astronomers to concentrate in scientific research.
- Foster reproducibility and maintainability. visualisation-aided data analysis often requires several parameter settings for pre-processing, plus the actual rendering of complex multidimensional datasets. Furthermore, in this era of Open Science, offering novel mechanisms and techniques to make scientific discoveries reproducible and maintainable is a must, especially for enhancing scientific and technical collaborations.
- Take advantage of a more flexible resource exploitation over heterogeneous HPC facilities, including also mixed HPC-Cloud resources (see also Section 3). Because of the increasing size and complexity of astrophysical datasets, there is a need for increasing the computing performances as well as the storage capacities for processing and analysis tasks while maintaining the cloud software-as-a-service opportunities.
- Minimize data-movement overheads and improve I/O performances. The importer modules of VisIVO rely on heavy I/O tasks for translating the astrophysical datasets to the internal VisIVO binary format which is used by the VisIVO filtering and visualisation modules. Therefore, minimizing the computing costs of these I/O tasks could potentially improve the overall performance of the VisIVO pipelines. This task will be addressed in the next release as outlined in section 2.3.4.

2.3.2 Tool development

In this section we outline the integration of VisIVO with a hybrid Workflows Management System (WMS) which allows its execution on multiple computing infrastructures including mixed HPC-Cloud resources, enabling parallelization of the workflow for exascale A&C simulations.

Our strategy consists of the integration of VisIVO with StreamFlow[26] [14] (see also Section 3), which addresses the need to improve portability by decoupling multiple components from platform-specific dependencies, while also fostering reproducibility and maintainability. Moreover, the proposed solution takes advantage of a flexible resource use over heterogeneous HPC facilities.

We use snapshots of cosmological N-body simulations performed with the OpenGadget3 and ChaNGa codes as input to the workflow for testing the integration of Streamflow with VisIVO with the aim of facilitating studies on the large-scale structure of the Universe.

Data For the development of the VisIVO tool we are currently using SPACE data from the OpenGadget3 and ChaNGa codes.

Additionally we have performed a scientific case study in [15] with large cosmological N-body simulations from the "Dark Energy and Massive Neutrino Universe" (DEMNUni) suite [16, 17]. We considered two different simulations based on the baseline *Planck* cosmology [18], specifically a flat ΛCDM model. The first simulation

[24] VisIVO, https://visivo.readthedocs.io/
[25] https://www.ivoa.net/documents/
[26] StreamFlow, https://streamflow.di.unito.it/

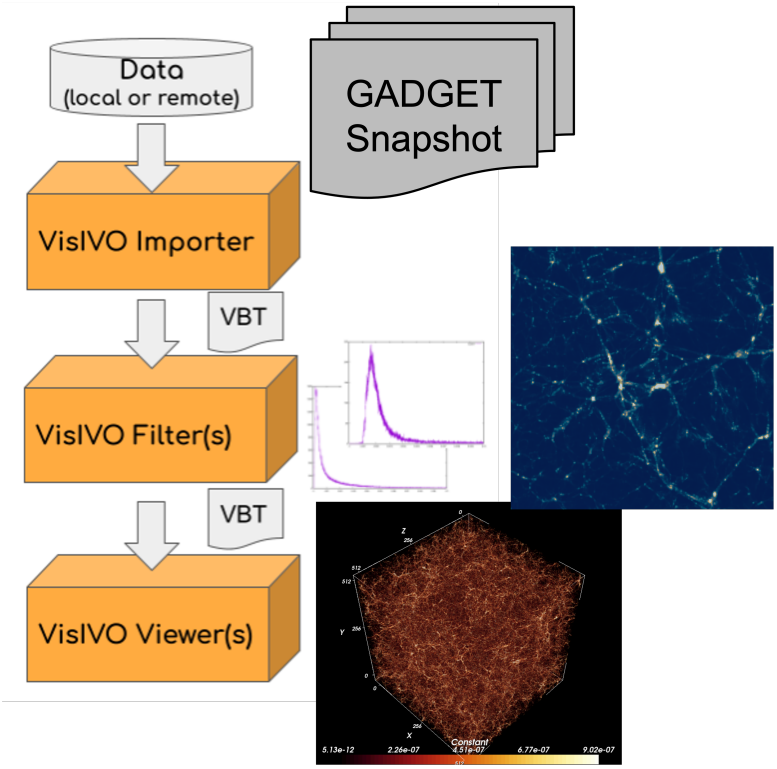


Figure 19: Typical visualisation pipeline in VisIVO (showing its three main modules) applied to OpenGadget3 cosmological simulation outputs.

follows the standard ΛCDM framework, while the second extends it to a $\nu\Lambda$CDM model by modifying only the total mass of the three active neutrinos to $M_\nu = \sum_i m_{\nu,i} = 0.16$ eV. In both cases, the matter density parameter $\Omega_m = 0.32$ and the amplitude of primordial curvature perturbations are kept fixed. Each simulation produced 64 outputs logarithmically equispaced in the scale factor $a \equiv 1/(1+z)$, within the redshift interval $z = 0 - 99$, including 50 between $z = 0$ and $z = 10$. For each of the 64 output times, a particle snapshot including all CDM and neutrino particles was produced on-the-fly.

Tools The VisIVO Server[27] is an application that allows to visualize a variety of large datasets generated by A&C simulations. It consists of three main modules for each of the following tasks: data importing, filtering, and viewing.

The importer module converts the supplied dataset into a VisIVO Binary Table (VBT), which is a highly efficient data representation internally used by the VisIVO Server. A VBT consists of two components: a header file (extension .bin.head) and a raw data file (extension .bin). The former consists of all necessary metadata, such as the overall number of fields and the number of points for each field (for particle datasets), or the number of cells and relevant mesh sizes (for volumetric datasets). The latter stores the actual data values sequentially, e.g. all elements of one field followed by all elements of the next field.

The data filtering module supports a variety of useful operations that can be performed on the VBTs, such as: randomization, decimation for reducing resolution, the application of mathematical or statistical functions, and cosmological post-processing. For the latter, three widely used mass assignment methods[19] can be employed to compute particle densities: nearest grid point (NGP), cloud-in-cell (CIC), and triangular-shaped cloud (TSC).

The visualisation process generates multi-dimensional views of the data, ensuring they fit within the available RAM. Supported visualisation types include data points, volumes, and vectors, all leveraging the capabilities of the visualisation Toolkit (VTK)[28].

In SPACE we are focusing on the implementation of new importer modules to handle data produced by the SPACE A&C codes and to tailor visualisation workflows to support the SPACE scientific cases.

[27] https://visivo.readthedocs.io/en/latest/
[28] VTK, https://vtk.org/

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

Figure 19 showcases the standard VisIVO visualisation pipeline designed for handling OpenGadget3 snapshots. A snapshot captures the state of all particles within the simulation box at a fixed point in time. The pipeline is structured around three core modules: `VisIVOImporter` ingests the data; `VisIVOFilter` computes particle densities; and `VisIVOViewer` generates the final visualisations.

We use workflow abstractions to provide a portable way to represent VisIVO modular applications and their resource requirements. This method improves reproducibility and maintainability while enabling users to efficiently exploit heterogeneous HPC facilities and reduce data transfer overhead. To achieve this, we integrated VisIVO with the StreamFlow workflow management system.

StreamFlow complements a standard workflow graph, composed of steps and their inter-dependencies, by adding a declarative description of potentially complex execution environments. This *hybrid workflow model* [20] enables the execution of workflow steps across multiple sites without requiring shared data spaces, authentication protocols, or bidirectional network channels. The StreamFlow control plane manages orchestration tasks such as data movement, fault tolerance, and provenance tracking.

With hybrid workflows, VisIVO modules can run in a multi-container environment, smoothly switching between different container runtimes (e.g., Docker, Singularity). Additionally, independent workflow steps can execute concurrently within distributed multi-agent ecosystems [21].

2.3.3 Results

Parallel VisIVO Importer for OpenGadget3 As a first result we implemented a novel VisIVO class, called `GadgetSource`, that handles the reading of OpenGadget3, i.e. snapshots in GADGET format and writing on a VisIVO Binary Table.

As a first step, each process needs to read data from the GADGET snapshot header, which is a special block placed at the beginning of the file. This is done by calling the `readHeader` method. The header contains multiple pieces of information necessary to read the rest of the file correctly, such as the number of elements for each type of particle in each file, the cumulative number for all the files, and the number of files into which the snapshot has been divided. By having the number of files, it is possible to call the `checkMultipleFiles` method, which verifies if all the files are available in the same folder as the input file. After this check, it is possible to read all the headers to obtain the information of each file by calling the `readMultipleHeaders` method.

After reading the header information, it is possible to use the `readData` method. Before starting the process of reading the binary data, each process has to create a list of blocks to read. This operation is necessary because each GADGET block might arrange data into tuples, and during the reading process, these tuples must be correctly arranged before writing them to the VBT file.

All the information regarding the blocks, such as the organization of data into tuples or the presence of a given block for a particular type of particle, is not available in the snapshot files. Therefore, we used simple unordered maps generated using information from the GADGET documentation.

This is a multi-process implementation using the Message Passing Interface (MPI) with a file-per-process approach, where each process reads different files to avoid file contention between processes. This is considered a common best practice in parallel I/O programming [22]. Additionally, the implementation does not require much communication between processes since all the needed information can be computed or read from the header. Therefore, the only form of communication needed is a barrier, where each process waits for the others to finish execution. Multi-process implementation in parallel I/O is fundamental for obtaining higher bandwidth with parallel file systems, such as BeeGFS [23].

The implementation is also multi-threaded, with threads spawned using OpenMP. With all the previously discussed information, each thread can compute the position and the file they need to read from and write to. There is no data contention because each thread reads and writes data at different positions. To avoid issues with the file descriptor and file pointer, each thread will manage its own file positions. The I/O operations used will be `pread()` and `pwrite()`, which are system calls suitable for multi-threaded applications as they allow I/O operations on the same file descriptor without being affected by changes to the file offset by other threads.

In Listing 1 there is a simplified version of the typical code executed by each thread, each thread has an exclusive couple of variables `nBlock` and `nFile`. In the first line the thread just computes the starting position in which the thread must write in the output file.

`typePosition` is an array that stores for each block the number of blocks preceding it, this is done for each particle type, and it is useful because it considers also multidimensional blocks as position or velocity.

`fileStartPosition` is an array that stores a number for each file corresponding to the number of particles handled by previous files, this is necessary to know the position from which to start writing. In line 3 `n` represents the number of iterations needed to handle each chunk of data.

Then we just have the parallel read and parallel write respectively at line 7 and 10.

Listing 1: GADGET VisIVOImporter thread code

```
1  startPos = typePosition[nBlock][type]* m_pHeaderType2[nFile].npartTotal[type] +
2  +fileStartPosition[type][nFile];
3  n=m_pHeaderType2[nFile].npart[type]/chunk;
4  for (int k = 0; k < n; k++)
5  {
6       bufferBlock = new float[blockSize*chunk];
7       pread(inputFileDescr[nFile], (char *)(bufferBlock), chunk*sizeof(float), offset);
8       offset += chunk*sizeof(float);
9       pWrite=((pToStart*sizeof(float)) + (k*chunk*sizeof(float)));
10      pwrite(outputFileDescr[type], (char *)(bufferBlock), chunk*sizeof(float), pWrite);
11 }
```

Preliminary scalability tests were conducted on multiple configurations, ranging from one to eight nodes and from one to 8 threads per node. For each configuration, we considered one MPI process per node. For these tests we used a dataset in GADGET format of size 512 GB.

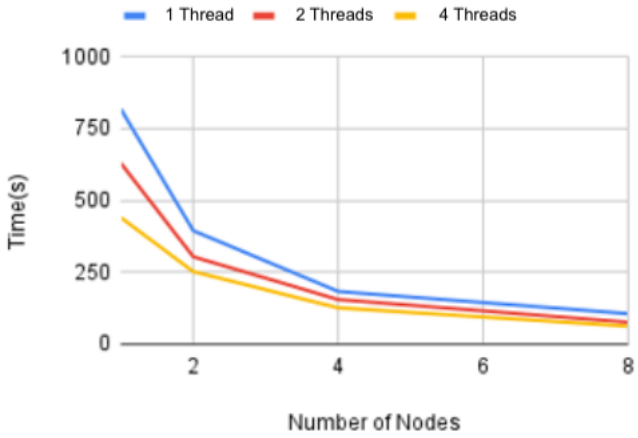


Figure 20: Preliminary scalability tests on 512GB Data.

In this preliminary test, as shown in Figure 20, we can see that multiple processes on multiple nodes can achieve an higher bandwidth, this behavior was expected on a parallel file system[23].

Multi-threading is limited by the number of blocks available in the simulation snapshots. Further testings will be performed as future works when additional sample files with more blocks will be available in SPACE.

VisIVO Importer for ChaNGa As for the second result we implemented a new class for VisIVO Importer to convert ChaNGa snapshots in Tipsy format into VisIVO Binary Tables, this class is called `ChangaSource`.

The first step for this new importer consists on reading the Tipsy header which consists of a structure composed by the time of the snapshot, the total number of total particles and the number of each type of particles, this information will be required in the successive step to read the particles data.

The second step consists of reading the particles, using the number of each type of particle and the format of each particle to adapt each field of the particles and write them in the right position of the VBTs. Tipsy snapshots can represent up to 3 types of particles which are: GAS, DARK and STAR, so this new class will generate up to 3 different VBT, one for each type of particle.

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

Implemented workflows The third main result is the use of the Common Workflow Language (CWL) to define our workflows[29] to visualize the GADGET and Tipsy snapshots. These CWL files are then used by StreamFlow to orchestrate the applications over heterogeneus computing infrastructures.

The workflows consist mainly of a sequence: `VisIVOImporter`, `VisIVOFilter`, and `VisIVOViewer` (Figure 19).

The end result is a particle rendering or a volume rendering provided by the VisIVOViewer module.

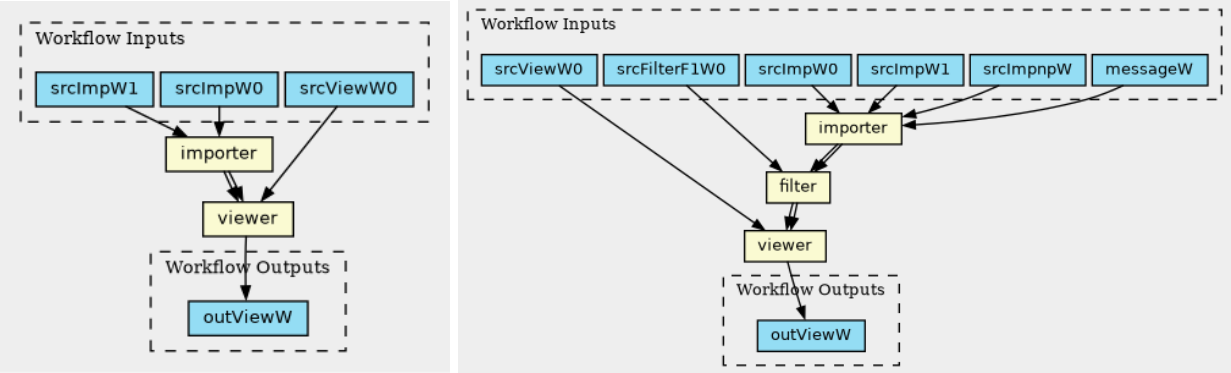


Figure 21: Visual representation of the VisIVO CWL workflows to generate particle renderings of ChaNGa output snapshots (on the left) and volume renderings of the OpenGadget3 output snapshots (on the right). The diagram lists all the inputs, outputs, and steps, and depicts the dependency relations between them. Workflows available at https://github.com/VisIVOLab/VisIVOCWL.

In the first workflow depicted in Figure 21 on the left the `VisIVOImporter` step executes the following script for the ChaNGa code output:

```bash
$ VisIVOImporter --fformat changa --out NewTable --file snapdir/snap_name
```

The script above converts a ChaNGa snapshot in Tipsy format and produces a new VBT for each type of particle in the snapshot. In our specific case, it produces three files, NewTableDARK.bin, NewTableGAS.bin, NewTableSTARS.bin and their corresponding header files, NewTableDARK.bin.head, NewTableGAS.bin.head and NewTableSTARS.bin.head. The header files contain information about the dark matter, gas components and stars of the input ChaNGa snapshot.

The second step corresponds to the viewer executing the following command:

```bash
$ VisIVOViewer --x POS_X --y POS_Y --z POS_Z --color --colorscalar RHO --colortable volren_glow --logscale
    --showbox --showaxes --showlut --out img NewTableGAS.bin
```

This command generates a particles rendering from the (NewTableGAS.bin, NewTableGAS.bin.head) VBT with the style features specified by the parameters, and saves it for multiple camera views using `.png` images.

In the second workflow depicted in Figure 21 on the right the `VisIVOImporter` step executes the following script for the OpenGadget3 code output:

```bash
$ export OMP_NUM_THREADS=AA
$ mpirun --np BB VisIVOImporter --fformat gadget --out NewTable --file snapdir/snap_name
```

The script above runs in parallel over `AA` OpenMP threads and `BB` MPI processes (where `AA` and `BB` are values passed as parameters to the workflow), so it can use multiple nodes if the infrastructures allows it. This command converts a GADGET snapshot and produces a new VBT for each type of particle in the snapshot. In our specific case, it produces two files, NewTableHALO.bin and NewTableGAS.bin, and their corresponding

[29] https://github.com/VisIVOLab/VisIVOCWL

header files, NewTableHALO.bin.head and NewTableGAS.bin.head. The header files contain information about the dark matter and gas components of the input GADGET snapshot.

In the next step of the workflow the filter module executes the command:

```bash
$ VisIVOFilter --op pointdistribute
    --resolution X_RES Y_RES Z_RES
    --points POS_X POS_Y POS_Z
    --out densityvolume.bin --file NewTableHALO.bin
```

This command takes as input the (NewTableHALO.bin, NewTableHALO.bin.head) VBT and executes the `pointdistribute` operation. This generates a table that contains the density values of the cells in the volume (calculated using the NGP, CIC, or TSC algorithms). The output is saved in the `densityvolume.bin` and `densityvolume.bin.head` files.

The last step corresponds to the viewer executing the following command:

```bash
$ VisIVOViewer --volume --vrendering
    --vrenderingfield Constant
    --color --colortable volren_glow
    --showlut --out img  --file densityvolume.bin
```

This command generates a volume rendering from the (densityvolume.bin, densityvolume.bin.head) VBT with the style features specified by the parameters, and saves it for multiple camera views using `.png` images.

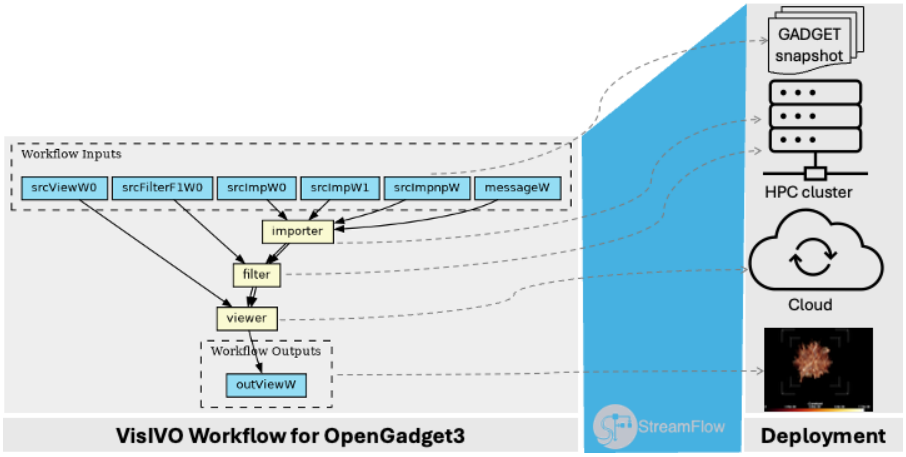


Figure 22: Sample deployment of a VisIVO Workflow applied to OpenGadget3 cosmological simulation outputs deployed on HPC and Cloud infrastructures.

The `.cwl` file describes the workflow using three other separate `.cwl` files, each describing and allowing the processing of a different step. The main CWL script defines a `Workflow` class, while each step within the workflow is implemented as a `CommandLineTool` class. VisIVO Docker containers are linked to their corresponding steps through the CWL `DockerRequirement` feature. The workflow steps with their inputs and outputs are represented in the dependency diagram depicted in Figure 21.

The workflows are then finally executed with StreamFlow which takes care of the deployment of each conteinerized VisIVO module executions on the different heterogeneous computing facility.

An example of the distributed workflow deployment with StreamFlow is available in Figure 22.

The `streamflow.yml` file is the entry point for StreamFlow and assigns the most appropriate execution environment to each workflow step, effectively integrating the hybrid layer into the workflow execution process. More details will be found in Section 3.1.3.

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

Rendering results Figure 23 shows some explificative renderings obtained from the VisIVO workflows depicted in Figure 21 on SPACE simulations output of the OpenGadget3 and the ChaNGa codes.

The final rendering produced for the DEMNUni simulations shows the time-evolution of the density of CDM particles as a movie[30] of the different snapshots described in the previous section. The left panel of Figure 24 shows the results of the volume rendering of the final snapshot of the simulation (corresponding to redshift $z = 0$). For a visual comparison, the right panel of Figure 24 shows a rendering of the density of the neutrinos in the simulation that includes neutrino particles.

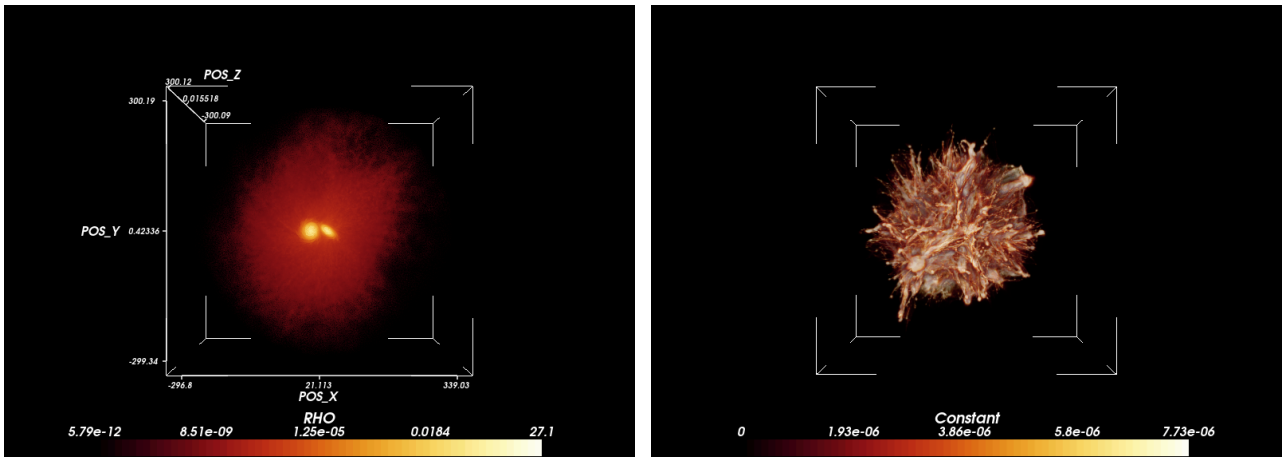


Figure 23: Examples of VisIVO workflow outputs. Particles rendering of GAS particles from ChaNGa (left panel) and from OpenGadget3 (right panel).

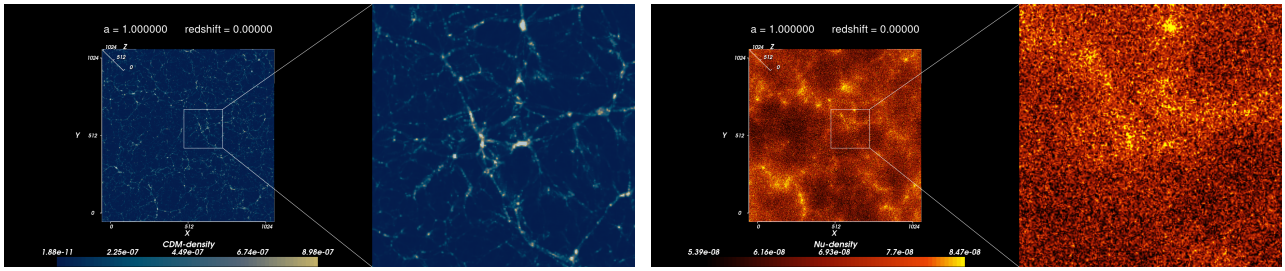


Figure 24: Examples of VisIVO workflow outputs of DEMNUni simulations. Volume rendering of CDM particles (left panel) and neutrino particles (right panel) at redshift $z = 0$.

2.3.4 Work plan for next release #2

Future works will aim at further testing the VisIVO workflows with other SPACE outcomes and scientific cases.

Furthermore, we aim to further expand the usage of VisIVO for the A&C community toward two directions: i) provide in-situ capabilities and ii) optimize the I/O between the different VisIVO modules employed in the workflows.

We are currently focusing on the integration with Hecuba (see Section 2.1) to allow in-situ visualisation with VisIVO to eliminate reliance on persistent storage, facilitating real-time analysis and visualisation during simulation execution. We foresee the development of a reader that retrieves data from a distributed database using Hecuba's API and the implementation of a complete pipeline that utilizes all VisIVO Server modules to generate visualisations from the data retrieved by the reader and passed in-memory through the VisIVO's filters and viewers.

In addition, we are evaluating technologies for the speed up of I/O between the different VisIVO modules exploring multiple optimization strategies and frameworks, such as CAPIO [24], to improve I/O performance by introducing streaming capabilities to workflow steps, see also Section 3.1.4 for additional details.

[30] https://youtu.be/rpLEZey3if4?si=ygSSLfnAuX5RcqND

Finally, we will investigate how to complement and enhance volume rendering visualisations comparing with the Blender developments and results presented in Section 2.2.

3 Workflows Use Case

3.1 Topology-aware workflows for modular Exascale data analysis

3.1.1 Introduction

This task aims to develop novel coordination semantics specifically designed to support the creation and execution of modular high-performance computing applications tailored for the next generation of Exascale computing systems. As we move towards increasingly complex and heterogeneous architectures, a new approach is required to effectively orchestrate and manage computational workloads across a wide range of computing resources. The proposed approach focuses on enabling topology-aware co-scheduling of loosely coupled components within complex scientific applications. This means that each component of a modular application will be intelligently allocated to the most appropriate part of the execution architecture, based on its specific resource needs and operational characteristics. For instance, components that are heavily dependent on input/output operations may be scheduled on nodes equipped with burst buffers to improve I/O performance, while computation-intensive tasks such as matrix operations could be directed to nodes featuring hardware accelerators like GPUs or FPGAs.

In this context, workflow abstractions play a crucial role by providing a high-level, portable, and modular representation of scientific applications. These abstractions describe the application in terms of atomic computational components, their resource requirements, and the data and control dependencies between them. Such an approach enhances not only portability but also the reproducibility[31], flexibility, and maintainability of complex applications across different platforms and configurations.

This task will investigate and develop the concept of topology-aware workflows. These are enhanced workflow representations that incorporate a high-level description of the resource topology, including node connectivity, memory hierarchies, and hardware specialisations. By enriching workflow graphs with this information, it becomes possible to enable more intelligent and adaptive orchestration strategies. These strategies aim to minimise costly data transfers, reduce contention, and exploit architectural heterogeneity to maximise performance and efficiency. Specifically, this task will define coordination semantics that allow the expression and optimisation of parallel workflow patterns commonly used in HPC application design, such as pipelining, task parallelism, and data parallelism. These semantics will serve as a foundation for more expressive and predictable workflow behavior, enabling both compile-time reasoning and runtime adaptation.

The Workflow Management System considered to orchestrate the described hybrid multi-HPC facility workflows is StreamFlow [32] framework[25], and this task will extend its capabilities. StreamFlow is an open-source, container-native workflow management system written in Python and based on the Common Workflow Language (CWL)[26]. StreamFlow will be enhanced to support the orchestration of modular HPC applications on Exascale platforms. These extensions will enable StreamFlow to leverage the proposed coordination semantics and topology-aware scheduling techniques, ultimately improving its ability to handle large-scale, distributed, and heterogeneous workloads efficiently.

3.1.2 Tool development

In this section, we briefly describe StreamFlow, which has been adopted in the EuroHPC ACROSS project and is currently in use within the EUPEX project. We also outline the extensions required to enable the orchestration of modular HPC applications on Exascale platforms.

StreamFlow is a container-native Workflow Management System (WMS) implemented in Python 3 and designed with two key principles: (i) support for multi-container execution (concurrent execution of multiple intercommunicating tasks within multi-container environments); and (i) decoupling from a shared data space (it removes the dependency on a single shared storage system, facilitating hybrid workflow execution across multi-cloud environments).

StreamFlow is compliant with the Common Workflow Language (CWL) standard, a language, by design, suitable for the description of modular workflows. It describes each step with explicit inputs and outputs.

[31] Reproducibility means that the scientific workflow can be reproduced by using the same dataset and software to obtain the same results.

[32] https://streamflow.di.unito.it/

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

Workflow steps in CWL are not necessarily run in the order they are listed, instead, the order is determined by the dependencies between steps. Workflow steps that do not depend on one another may run in parallel. Together with StreamFlow, it assures the portability and reproducibility of the considered workflow.

StreamFlow is by itself designed in a modular way, i.e, to connect with different executing platforms, it is enough to use the corresponding connector. Its structure can be seen in Figure 25, where different connectors and plugins are listed, such as for Slurm, Docker, Flux and etc. The plugin for the Microsoft Azure cloud platform is currently in design.

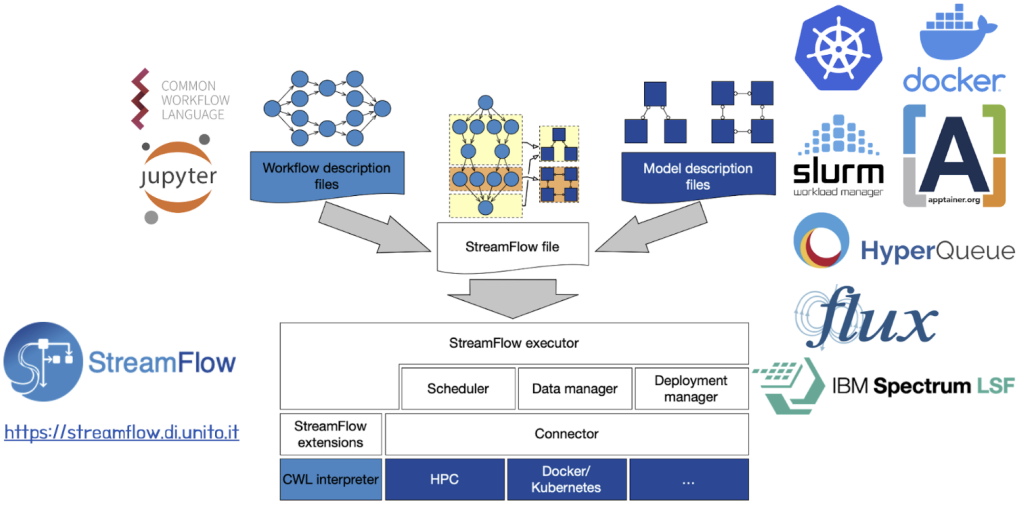


Figure 25: StreamFlow Workflow Management System

Extensions of StreamFlow in the context of the SPACE project:

- Thanks to StreamFlow's modular design and plugin architecture, users can implement custom behaviors and experiment with new workflow orchestration ideas. The StreamFlow codebase already includes several connectors that enable communication with different environments and interoperability with external tools. Additional connectors, such as the LSFConnector available on PyPI, are distributed as external plugins. Recently, leveraging this extensibility, we developed a new connector called `TwoFaConnector` [33], which enables SSH connections to locations that require two-factor authentication. An example of such an infrastructure is the Slovenian HPC facility Vega[34].
- Since many HPC facilities are accessible via SSH, having a robust and resilient `SSHConnector`[35] in StreamFlow is crucial. To this end, we enhanced the existing retry mechanism to handle additional failure scenarios.
- The xFFL framework has been significantly improved to support flexible customization of execution configurations, such as specifying which HPC systems participate in the workflow, selecting different training scripts, and other runtime parameters [36].

3.1.3 Results

In this section, we report the two main results: the exploitation of the StreamFlow with the VisIVO workflows and the first prototype of the hybrid and heterogeneous cross-facility execution modeled in CWL and orchestrated by StreamFlow.

[33] https://github.com/alpha-unito/streamflow-twofa
[34] https://izum.si/en/vega-en/
[35] https://github.com/alpha-unito/streamflow/commit/d3b82ecac0a68f59a998ea6d04527cd115f8adb8
[36] https://github.com/alpha-unito/xffl/commit/4bfe7729a30bbfac4e0448d33ec4b25ebf226811

```yaml
version: v1.0
workflows:
  visivo:
    type: cwl
    config:
        file: workflow.cwl
        settings: config.yml
    bindings:
        - step: importer
          target:
            deployment: HPC
        - step: filter
          target:
            deployment: HPC
        - step: viewer
          target:
            deployment: Cloud
```

```yaml
deployments:
  HPC:
    type: #slurm
    config:
      file: #path/to/HPC-sbatch.sh
      hostname: #address
      maxConcurrentSessions: #number of concurrent sessions
      sshKey: #sshKey
      username: #username
    workdir: #path/to/the/workdir
  Cloud:
    type: #ssh
    config:
      file: #path/to/.sh file
      nodes:
        - worker-1
      sshKey: #sshKey
      username: #username
```

Figure 26: The `streamflow.yml` file, template for deploying two steps on HPC and Cloud environment.

Visualization Results Using StreamFlow and VisIVO The VisIVO workflows presented in Section 2.3 are executed using the following command with StreamFlow, where the `.yml` file specifies the input parameters (including the `.cwl` file that defines the workflow and the `config.yml` file that describes the input configuration).

```bash
$ streamflow run streamflow.yml
```

The `streamflow.yml` file (the entry point for StreamFlow) assigns the most appropriate execution environment to each workflow step, effectively integrating the hybrid layer into the workflow execution process. For instance, the `streamflow.yml` file in Figure 26 represents the template file for deployment of the three steps of the VisIVO workflow: importer and filter (deployed on HPC cluster) and viewer (deployed on cloud). The mapping of workflow steps to their corresponding environments is specified in lines 1-17, while the deployment section, defined in lines 18–35, describes the environments themselves.

Initially, StreamFlow translates the CWL dataflow semantics into an internal workflow representation, explicitly constructing a macro dataflow graph [27]. To ensure flexibility, StreamFlow employs a three-layer hierarchical model for representing execution environments:

- a `deployment` is the *unit of allocation*, i.e., all its components are always co-allocated when executing a step;
- a `service` constitutes the *unit of binding*, i.e., StreamFlow users can map each workflow step to a single service for execution;
- a `resource` is a single instance of a potentially replicated service and constitutes the *unit of scheduling*, i.e., each step of a workflow is offloaded to a configurable number of resources to be processed.

StreamFlow to orchestrate heterogeneous cross-facility workflow execution As a prototype for applying the CWL as the coordination semantics for modular HPC applications targeting heterogeneous, distributed Exascale systems, we proposed xFFL: Cross-Facility Federated Learning [28]. The xFFL experiments are specifically designed to explore how large-scale AI models can be efficiently deployed across single or multiple HPC systems, with the overarching aim of defining scalable deployment methodologies.

One of the main challenges in orchestrating cross-facility workflow execution is connecting to different HPC centers and cloud clusters. Some require SSH connections, others rely on various privacy certificates, and some implement two-factor authentication procedures, among other methods. Consequently, to enable seamless automated access, it becomes necessary to develop different plugins for StreamFlow tailored to each authentication and access mechanism (described in section 3.1.2).

To test CWL and StreamFlow in a hybrid, cross–HPC center execution, we conducted two preliminary experiments:

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

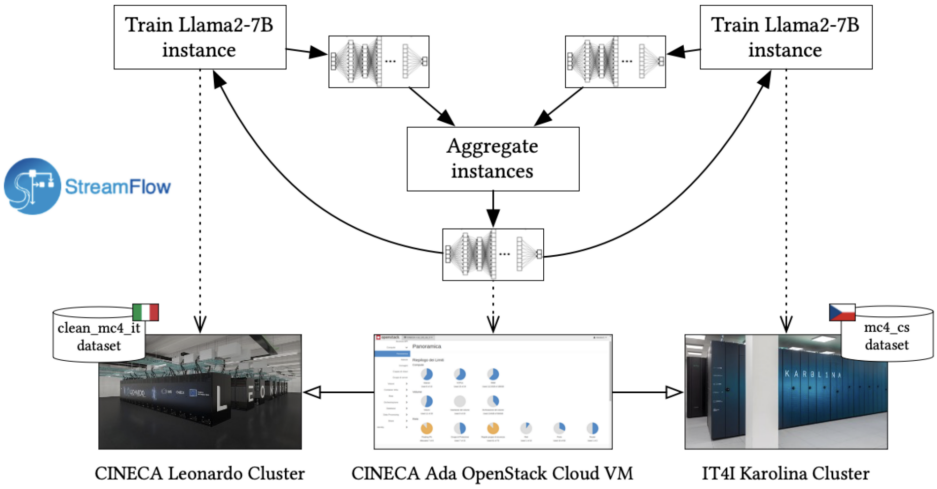


Figure 27: xFFL: Cross-Facility Federated Learning.

- We exploited two EuroHPC supercomputers cooperatively: Leonardo (CINECA, Bologna, Italy) and Karolina (IT4Innovations, Ostrava, Czech Republic) to train the LLaMA 2 (7 billion parameters) model using the protocol illustrated in Figure 27. StreamFlow coordinates Slurm jobs, each training an aggregated LLaMA model across multiple supercomputers. After training, the orchestrator collects the models for aggregation. Notably, datasets remain local to each site, preserving data privacy.
- We extended the setup to three EuroHPC supercomputers cooperatively: Leonardo (CINECA, Bologna, Italy), LUMI (CSC, Kajaani, Finland), and MeluXina (LuxProvide, Bissen, Luxembourg) to train LLaMA-3 (8 billion parameters).

Although this work is primarily demonstrated in the context of Federated Learning models, its applicability extends well beyond that domain. The proposed approach is broadly adaptable to any complex application that requires modular workflows operating across heterogeneous and distributed environments. This effort marks an important first step toward enabling scalable and modular workflow execution for workloads in the SPACE domain.

3.1.4 Work plan for next release #2

For future work, we have two main directions:

- As the use of computational resources continues to grow, so does the need to build resilient systems. A part of this task will involve exploring and enhancing the resilience capabilities of the StreamFlow framework by extending the hybrid multi-facility execution to include the Vega HPC center.
- While testing the VisIVO workflows coordinated by StreamFlow, we identified opportunities to enhance performance by optimizing VisIVO's I/O operations. The approach involves leveraging the streaming capabilities provided by CAPIO: Cross-Application Programmable I/O [24, 29] within the VisIVO workflow.

4 Conclusions

The first release of the SPACE CoE High Performance visualisation tools reflects the advancements toward the development of high-performance visualisation tools and techniques tailored for astrophysical and cosmological research, as outlined in the SPACE objectives. Specifically, it targets the three critical use cases in visualisation and one in workflows identified in the previous deliverable D3.1: In-situ Parallel visualisation through Hecuba; Interactive, volume-based cinematic visualisation; Interactive, distributed visualisation for High Performance Computing and cloud environments; Topology-aware workflows for modular Exascale data analysis. Moreover,

the developments have been consistent with the thorough planning and documentation process carried out during the first year, which was reported in the aforementioned deliverable D3.1.

The objectives covered by these use cases are strategically chosen because of their relevance for the A&C community, but also because they address complex challenges posed by the scale and complexity of cosmological simulation data in the exascale era, aiming to facilitate improved interpretability, computational efficiency, and scientific accuracy.

In the current stage of development, the four use cases reported here meet the requirements of this #1 release as stated in Milestone M7 of being "provisional but fully operational". The work reported in this document constitutes a solid base in the areas covered by our use cases towards the improvements and future developments that will take place during the last timespan of SPACE.

This document is Public (PU)
and was produced under SPACE project EU GA 101093441

References

[1] "Blender," 2025. [Online]. Available: https://www.blender.org/

[2] "CyclesPhi," 2025. [Online]. Available: https://code.it4i.cz/raas/cyclesphi

[3] K. Museth, "Vdb: High-resolution sparse volumes with dynamic topology," *ACM Trans. Graph.*, vol. 32, no. 3, Jul. 2013. [Online]. Available: https://doi.org/10.1145/2487228.2487235

[4] Academy Software Foundation, "Openvdb," https://github.com/AcademySoftwareFoundation/openvdb, 2025, accessed: 2025-06-26.

[5] N-Body Shop, "Nchilada file format," https://github.com/N-BodyShop/utility/wiki/NChilada-File-Format, 2025, accessed: 2025-06-26.

[6] K. Museth, "Nanovdb: A gpu-friendly and portable vdb data structure for real-time rendering and simulation," in *ACM SIGGRAPH 2021 Talks*, ser. SIGGRAPH '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3450623.3464653

[7] M. Jaros, P. Strakos, and L. Tornatore, "Volume-based cinematic visualization from OpenGADGET," Poster presented at HPCSE2024, 2024.

[8] G. Taffoni, A. Mignone, L. Tornatore, E. Sciacca, M. Guarrasi, G. Lapenta, L. Riha, R. Vavrik, O. Vysocky, K. Kadlubiak, P. Strakos, M. Jaros, K. Dolag, B. Commercon, L. Rezzolla, K. Pierre, G. Doulis, S. Shen, M. Marazakis, D. Gregori, E. Boella, G. Perna, M. Zanotti, E. Raffin, K. Polsterer, S. T. Gomez, and G. Marin, "Redesign of astrophysical codes for exascale computing: the SPACE experience," in *Software and Cyberinfrastructure for Astronomy VIII*, J. Ibsen and G. Chiozzi, Eds., vol. 13101, International Society for Optics and Photonics. SPIE, 2024, p. 131010X. [Online]. Available: https://doi.org/10.1117/12.3020361

[9] 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, and R. Wissing, "High Performance Visualization for Astrophysics and Cosmology," in *Proceedings of PDP2025*, 2025.

[10] S. Zellmann, M. Jaros, J. Amstutz, and I. Wald, "GPU Volume Rendering with Hierarchical Compression Using VDB," in *Proceedings of EGPGV2025*, 2025.

[11] Y. Patel, S. Appalaraju, and R. Manmatha, "Human Perceptual Evaluations for Image Compression," Aug. 2019.

[12] F. Vitello, E. Sciacca, U. Becciani, A. Costa, M. Bandieramonte, M. Benedettini, M. Brescia, R. Butora, S. Cavuoti, A. M. Di Giorgio *et al.*, "Vialactea visual analytics tool for star formation studies of the galactic plane," *Publications of the Astronomical Society of the Pacific*, vol. 130, no. 990, p. 084503, 2018.

[13] G. Tudisco, F. Vitello, E. Sciacca, S. Riggi, S. Molinari, E. Malikova, and M. Krokos, "Vialactea: a distributed visual analytic system for exploring our galactic ecosystem," in *Astronomical Society of the Pacific Conference Series*, vol. 535, 2024, p. 211.

[14] I. Colonnelli, B. Cantalupo, I. Merelli, and M. Aldinucci, "StreamFlow: cross-breeding cloud with HPC," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1723–1737, 2021.

[15] E. Sciacca, V. Cesare, N. Tuccari, F. Vitello, I. Colonnelli, C. Carbone, E. Tramontana, and U. Becciani, "Toward a portable and reproducible high- performance visualization for astronomy & cosmology," Sep. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.13858498

[16] C. Carbone, M. Petkova, and K. Dolag, "DEMNUni: ISW, Rees-Sciama, and weak-lensing in the presence of massive neutrinos," *Journal of Cosmology and Astroparticle Physics*, vol. 2016, no. 7, p. 034, Jul. 2016.

[17] G. Parimbelli, C. Carbone, J. Bel, B. Bose, M. Calabrese, E. Carella, and M. Zennaro, "DEMNUni: comparing nonlinear power spectra prescriptions in the presence of massive neutrinos and dynamical dark energy," *Journal of Cosmology and Astroparticle Physics*, vol. 2022, no. 11, p. 041, Nov. 2022.

[18] Planck Collaboration, P. A. R. Ade *et al.*, "Planck 2013 results. XVI. Cosmological parameters," *Astronomy & Astrophysics*, vol. 571, p. A16, Nov. 2014.

[19] W. Cui, L. Liu, X. Yang, Y. Wang, L. Feng, and V. Springel, "An ideal mass assignment scheme for measuring the power spectrum with fast fourier transforms," *The Astrophysical Journal*, vol. 687, no. 2, p. 738, 2008.

[20] I. Colonnelli, "Workflow models for heterogeneous distributed systems," in *Proceedings of the 2nd Italian Conference on Big Data and Data Science (ITADATA 2023), Naples, Italy, September 11-13, 2023*, ser. CEUR Workshop Proceedings, N. Bena, B. D. Martino, A. Maratea, A. Sperduti, E. D. Nardo, A. Ciaramella, R. Montella, and C. A. Ardagna, Eds., vol. 3606. CEUR-WS.org, 2023.

[21] R. F. Da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman, "A characterization of workflow management systems for extreme-scale applications," *Future Generation Computer Systems*, vol. 75, pp. 228–238, 2017.

[22] S. L. Sandra Mendez, *Best Practice Guide - Parallel I/O*. Partnership for Advanced Computing in Europe, 2019.

[23] L. T. Francieli Boito, Guillaume Pallez, "The role of storage target allocation in applications' i/o performance with beegfs," *CLUSTER 2022 - IEEE International Conference on Cluster Computing, Heidelberg, Germany.*, 2022.

[24] A. R. Martinelli, M. Torquati, M. Aldinucci, I. Colonnelli, and B. Cantalupo, "CAPIO: a middleware for transparent I/O streaming in data- intensive workflows," in *2023 IEEE 30th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2023, pp. 153–163.

[25] I. Colonnelli, B. Cantalupo, I. Merelli, and M. Aldinucci, "StreamFlow: cross-breeding cloud with HPC," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1723–1737, 2021. [Online]. Available: https://doi.org/10.1109/TETC.2020.3019202

[26] M. R. Crusoe, S. Abeln, A. Iosup, P. Amstutz, J. Chilton, *et al.*, "Methods included: standardizing computational reuse and portability with the common workflow language," *Commun. ACM*, vol. 65, no. 6, p. 54–63, 2022.

[27] I. Colonnelli, B. Cantalupo, R. Esposito, M. Pennisi, C. Spampinato, and M. Aldinucci, "HPC application cloudification: The StreamFlow toolkit," in *12th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and 10th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2021)*, ser. Open Access Series in Informatics (OASIcs), J. Bispo, S. Cherubin, and J. Flich, Eds., vol. 88. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, p. 5:1–5:13.

[28] I. Colonnelli, R. Birke, G. Malenza, G. Mittone, A. Mulone, J. Galjaard, L. Y. Chen, S. Bassini, G. Scipione, J. Martinovič, V. Vondrák, and M. Aldinucci, "Cross-facility federated learning," *Procedia Computer Science*, vol. 240, pp. 3–12, 2024, proceedings of the First EuroHPC user day. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050924016909

[29] M. E. Santimaria, A. R. Martinelli, I. Colonnelli, B. Cantalupo, M. Torquati, and M. Aldinucci, "Capio-cl: The capio coordination language," *International Journal of Parallel Programming*, vol. 53, no. 2, p. 10, 2025. [Online]. Available: https://doi.org/10.1007/s10766-025-00789-0

This document is Public (PU)
and was produced under SPACE project EU GA 101093441