Continuous simulation data stream: a dynamical timescale-dependent output scheme for simulations

Hausammann, L.; Gonnet, P.; Schaller, M.

Astronomy and Computing 41 (2022) 100659

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

Continuous Simulation Data Stream: A dynamical timescale-dependent output scheme for simulations

L. Hausammann [a,*], P. Gonnet [b], M. Schaller [c,d]

[a] Laboratoire d'Astrophysique, Ecole Polytechnique Fédérale de Lausanne (EPFL), 1290 Sauvergny, Switzerland
[b] Google Switzerland, 8002 Zürich, Switzerland
[c] Lorentz Institute for Theoretical Physics, Leiden University, PO Box 9506, NL-2300 RA Leiden, The Netherlands
[d] Leiden Observatory, Leiden University, PO Box 9513, NL-2300 RA Leiden, The Netherlands

ARTICLE INFO

Article history:
Received 2 April 2022
Accepted 30 September 2022
Available online 7 October 2022

Keywords:
Methods: numerical
Software: simulations
Simulations: I/O
Galaxies: evolution

ABSTRACT

Exa-scale simulations are on the horizon but almost no new design for the output has been proposed in recent years. In simulations using individual time steps, the traditional snapshots are over resolving particles/cells with large time steps and are under resolving the particles/cells with short time steps. Therefore, they are unable to follow fast events and use efficiently the storage space. The Continuous Simulation Data Stream (CSDS) is designed to decrease this space while providing an accurate state of the simulation at any time. It takes advantage of the individual time step to ensure the same relative accuracy for all the particles. The outputs consist of a single file representing the full evolution of the simulation. Within this file, the particles are written independently and at their own frequency. Through the interpolation of the records, the state of the simulation can be recovered at any point in time. In this paper, we show that the CSDS can reduce the storage space by 2.76x for the same accuracy than snapshots or increase the accuracy by 67.8x for the same storage space whilst retaining an acceptable reading speed for analysis. By using interpolation between records, the CSDS provides the state of the simulation, with a high accuracy, at any time. This should largely improve the analysis of fast events such as supernovae and simplify the construction of light-cone outputs.

1. Introduction

With the arrival of the era of exa-scale computing, scientists across all domains have been focused on improving the performances of their software. They have been mostly looking at the scaling of their own physical computation (examples in astrophysics include Schaller et al., 2016; Jetley et al., 2008; Potter et al., 2017; Adams et al., 2009; Müller et al., 2019) with, so far, little attention given to the way of writing the outputs. Indeed, in recent years, the HDF5 format (The HDF Group, 1997) has become the de-facto standard with some applications also relying on custom-made binary files (e.g. Maksimova et al., 2021; Potter et al., 2017). Scientists rely often solely on the improvements within the HDF5 library itself or on the hardware for their own code. This approach is generally successful with simulations writing data in parallel from 1000s of inter-connected nodes. This format is particularly well adapted for snapshots that simply write the state of the simulation at a few discrete times (e.g. Nelson et al., 2015; Norman et al., 2007) containing up to trillions of particles (Potter et al., 2017; Maksimova et al., 2021; Bocquet et al., 2016). Whilst some studies or software developments have been focused on improving the performances of the I/O (e.g. Xiao et al., 2012; Ross et al., 2008; Ma et al., 2006; Mitra et al., 2005; Ragagnin et al., 2017), or on increasing the level of abstraction (e.g. Godoy et al., 2020; Lüttgau et al., 2018; Zheng et al., 2013; Abbasi et al., 2009), little work has been done on rethinking the general framework used to store data from simulations.

An interesting case where such rethinking was done is the construction of light-cone data. Such outputs reproduce the behavior of observations, where looking further away corresponds to looking back in time. As only the particles contained within the light cone surface are interesting, techniques have been developed to increase the performance of the simulation (e.g. Garaldi et al., 2020) and to write the outputs in a more efficient way (e.g. Evrard et al., 2002). This last technique consists in writing a type of snapshot where the particles are written only when they cross the light cone surface. While this approach needed to rethink the outputs, it is not a general solution for astrophysics.

In cosmological simulations (but also more generally in other simulations where many time-scales are coupled), the gravity is producing large difference of time scales through the simulated volume. As it would require too much computational time to use a single, global, time-step size, a multi (local) and individual time

* Corresponding author.
E-mail address: loic.hausammann@protonmail.com (L. Hausammann).

https://doi.org/10.1016/j.ascom.2022.100659

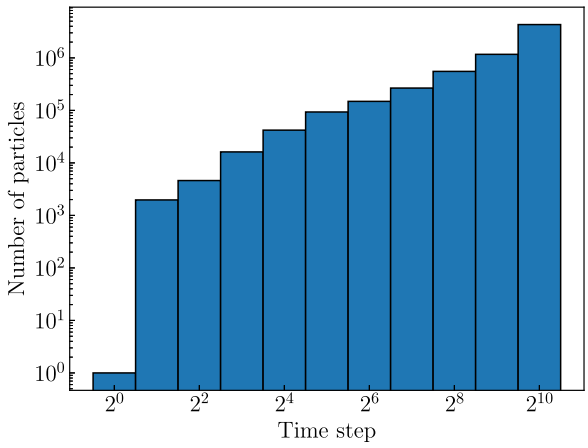

Fig. 1. Distribution of time-step sizes (normalized to the smallest time-step) extracted from a cosmological simulation run with the EAGLE model within a periodic box of 12 Mpc at $z = 3$. In this simulation, the ratio between the largest and smallest time steps is already above 1000 and only a small fraction of the particles really need a high time resolution.

step approach was designed where each particle or cell is evolved at its own time scale. This approach ensures that all the particles or cells are evolved at the same relative accuracy (Aarseth, 1963; Springel, 2005). As an example of this, in Fig. 1, we show the distribution of time steps within a cosmological simulation run with the EAGLE model (Schaye et al., 2015). Only an almost negligible fraction of the particles actually needs a small time step and the simulation analysis often focuses on them. They are located in the most active regions of the simulation Even if this example simulation is relatively small (10^7 particles), already a ratio of 1000 can be observed between the smallest and largest time-step sizes. For the outputs, it means that writing all the particles/-cells together can potentially provide a far too high accuracy for particles/cells within regions using long time-steps (e.g. in voids) and far too low for particles/cells using very short steps (e.g. within galaxies). A consequence of this is that the traditional snapshot model is unable to follow accurately fast events such as supernovae feedback effects and render their interpretation more complicated. Only by writing an impossibly large number of snapshots could such events be properly followed in large-scale simulations. To remedy this issue, many codes are relying on an additional set of files that describes such fast events (e.g. writing all the supernovae into a file). While this approach is sufficient in many cases, it lacks a complete description of the event's environment and its impact through time. It does also require an a priori knowledge of what events will be of interest.

In this paper, we present a novel approach, the *Continuous Simulation Data Stream* (CSDS), which allows us to recover the state of the simulation at any time with higher accuracy and a lower storage cost than snapshots. To simplify the discussion, in this paper, we will focus on particle based codes, but this method could certainly be adapted for Adaptive Mesh Refinements (AMR) or to any other general method. In practice, the CSDS is implemented as a stand-alone module around the open-source cosmological code SWIFT[1] (Schaller et al., 2016, 2018).

The paper is organized as follows. In Section 2, we describe in details the theory behind the CSDS. In Section 3, the implementation within SWIFT is provided followed by Section 4 where the reading strategy is explained. Our results are separated in two Sections (5 and 7). The first one presents the efficiency of the CSDS and the second one shows some examples of applications.

[1] www.swiftsim.com.

The implementation presented in this paper is fully open-source and included as part of the SWIFT repository.[2]

2. The Continuous Simulation Data Stream

The Continuous Simulation Data Stream (CSDS) is a new output strategy and mechanism aimed at replacing or supplementing, at least partially, the traditional snapshot system. Its main advantage resides in the writing of each individual resolution element in their own mini logbook. It hence allows to capture their evolution according to their *own time-scale* and not only at the fixed time resolution set globally by traditional snapshots. Thus, it perfectly adapts to simulations using individual localized time-step sizes and can achieve extremely high time resolution while keeping the output size reasonable. Capturing the detailed progression of the elements using the smallest time-step bin would require a prohibitive number of fixed-time snapshots; whilst our mechanism allows us to track this evolution precisely.

2.1. Overview of the different components

To achieve this high resolution, the CSDS uses a single file, called the *logfile*, that describes the whole evolution of the simulation. Within this file, all the information is written in the form of per-particle *records* (except for the header containing some general meta-data). This approach allows to write the particles individually and at their own timescale without any need to synchronize them during the simulation. When reading the CSDS for analysis, a synchronization is still required and done through an interpolation.

To construct the logfile, a general header is written during the initialization of the simulation. Then, a time record for the initial step is written followed by a particle record for all the particles in the volume (i.e. this corresponds to the initial conditions). The particles then carry a clock (see below) giving them an individual dump frequency. At each step of the simulation, the CSDS writes a *time record* and a *particle record* for the particles whose timer is up. At the end of the simulation, all the particles are written one final time and a time record concludes the logfile. Users can hence reconstruct the entire history of a given particle by hoping from record to record through the logfile. If a user requests details about a given particle in-between two time-records, interpolation is used. The accuracy (faithfulness) of the whole mechanism is entirely dictated by the frequency of the dump of each particle. We note, for instance, that for the simulation shown in Fig. 1, a large fraction of particles would not need any records between their initial and final dump as their trajectories can be well recovered just by interpolation.

To simplify the reading of the logfile, the CSDS uses a set of *index files* that describe the state of the simulation at a given time. While this mimics the behavior of snapshots, they tend to be smaller as they only contain, for each particle, the last known location within the logfile and IDs of the particles. The index files contain also the history of the particles created and deleted during the simulation (e.g. for `MPI` exchanges, star formation and black hole interactions for cosmological simulations). These files are generated after the simulation and, from our experience, a small number ($\sim$ 10) of index files is sufficient to achieve good analysis performance.

[2] For the *csds-reader* we used version 1.5 (available at https://gitlab.cosma.dur.ac.uk/lhausammann/csds-reader) in this paper. More specifically, all the plots and numbers shown in this paper were obtained using commit `d078e2dd` of SWIFT and commit `c073baf0` of the reader code.

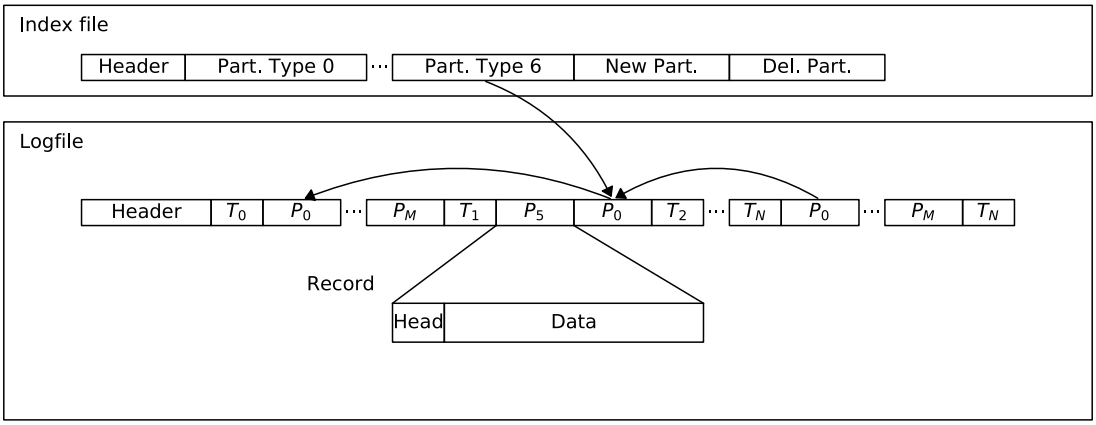

Fig. 2. Representation of the different files. The logfile contains the output of the simulations and the index files can be used to speed up the reading of the logfile. The logfile starts with a header containing information about the format (e.g version numbers and the different masks, see Section 2.3). Then the file will only contain records such as the one in the zoom-in. They can either contain a time or a particle and are always composed of a header and the data. In the header, a mask is stored in order to describe the data part and also an offset to the previous or next corresponding record (see Section 2.2). At the beginning of the simulation, we first write down the time (T_0) and the initial conditions (P_0 to P_M). Until the end of the simulation, the CSDS writes down the time step at the beginning of each step and only the required active particles (e.g. P_0 and P_5, see Section 2.6 for more information about the writing criterion). Finally, the code writes the particles at the final time and ends up with a second writing of the time step as a sentinel indicating the end of file. The index files are written once the simulation is done in order to speedup the reading. They start with a header (e.g. simulation time, number of particles, etc.) and then the particles sorted by type. For each particle, the position of the last record (offset) and their ID is written. At the end, the file contains the information about the particles created and removed since the last index file. They are still written in the form of an offset and an ID.

Finally, we also include a *metadata file* that contains details on the simulation (e.g. units, cosmological model, subgrid parameters). This file will not be discussed further as the details are not important to the CSDS mechanism.

As a summary, the different objects used in the CSDS and their relations are shown in Fig. 2.

2.2. Description of the records

Each record in the logfile is made of a small header followed by the data, as shown in the zoom-in at the bottom of Fig. 2. In the header, we store a *mask* and an *offset*.

The mask is a simple bit mask describing the fields contained in the data (coordinates, velocities, internal energies, etc.). This means that based only on the bit mask, a record can be identified as containing either the current time or a particle field. It also means that we can have different writing frequencies for different particle fields (e.g. metallicities evolve much more smoothly than temperatures, effectively requiring fewer entries). The offset is the distance in the file to the *previous* record of the particle. It means that the evolution of a particle can be quickly followed backwards thanks to this offset.

As usually we are interested in moving forward in time, the offset of the records are reversed during the first read of a logfile after a simulation has completed.[3]

In the data part of a record, we simply copy the fields corresponding to the mask into the file. As the different fields are written the one after the other without any information in between, the order when writing and reading a record must be respected. This order is indicated in the logfile header.

2.3. Description of the logfile

The logfile starts with a header that contains the version, the direction of the offset (in order to know if they need to be reversed), the size of the strings, the number of different masks, all the available masks (name and data size), and finally the masks for each particle types (in the writing order). It could be extended with more data without risk as we also store the position of the first record. This general structure is shown in Fig. 2.

When the simulation starts, the initial time is written followed by a record for each particle presents in the initial conditions. As the CSDS requires some interpolations, this step is required in order to avoid any extrapolation. Then at each simulation step, the CSDS writes a time record followed by as many particle records as required by their individual write frequencies (see Section 2.6 for a discussion about the writing criterion). The fraction of particles written during a single step can largely differ between different steps. On some steps, the CSDS writes 0 particle due to a low number of active particles and/or a lack of active particles requiring a log. The opposite case also exists where all the particles are written. At the end of the simulation, we write the final time followed by all the particles in order to avoid any extrapolation and conclude with a copy of the last time record as a sentinel marking the end of the file.

2.4. Restart strategy

HPC software should expect to be killed at any time due to a system failure or to be limited in the amount of time they can run. In order to restart the simulation, different strategies exist, but all relies on dumping at least part of their memory into so-called check-point files and restarting from them.

The CSDS is no different from the rest of the software package in this situation. While the simulation would be written to the check-point files as usual, the CSDS simply dumps the position of the last record written in the logfile. When restarting, the CSDS starts writing after the position of the last record and thus automatically get rid of the unnecessary data written between the last restart file and the crash. We emphasize that we do not intend the CSDS mechanism to be used as a check-pointing mechanism by itself.

2.5. Description of the index file

The index files are totally optional as they contain information that exists in the logfile. They, however, allow for an efficient usage of the logfile. For example, if the simulation at the final time

[3] This operation is one of the slowest. Thus we might drop it in the future and encourage the users to move backward in time.

is requested, without the index files, it would be required to read the whole logfile in order to find all the existing particles and then update them until reaching the final time. To solve this problem, the index files contain enough information to reconstruct quickly the state of a simulation *at a given time* (like a traditional snapshot would) using the logfile.

The index files are generated when reading the logfile and are set at regular interval in simulation time.

An index file starts with the current time step and number of particle and then an array of ID and offset to the last record is written for each particle (sorted by particle's type). Next, we write down the history of the created and suppressed particles since the last index file (if any). The two histories are built in the same way than the current state. They consist first in the number of new particles along with an array of ID and offset (sorted by type) and then the same for the suppressed particles.

It is worth mentioning that a low number of index files requires a larger amount of random access memory (RAM) than a large number of index files. Indeed, the history of all the particles created or removed in between two index files has to be kept in memory. So increasing the number of index files can help on machines with a low amount of memory.

2.6. Particle writing criterion

The CSDS writes down particles at different times and uses interpolations to reconstruct a snapshot at any time in-between records. A natural consequence of this is that the criterion for writing a particle should depend on the required quality for the reconstruction.

A possibility is to keep in memory some information about the particle since the last record and use this information to evaluate the quality of the future interpolation. For example, if one wishes to do a linear interpolation of a field f between records, the error ($\propto \mathrm{d}t^2 \max |f''(t)|$) could be evaluated based on the time difference since last record (dt) and the second derivative of f. This method works well with the CSDS as it can be applied independently to each particle field and then write only a subset of the fields in each record. It would therefore reduce the size of the logfile without losing accuracy. This is also philosophically very desirable. It means users can specify the accuracy of the desired output rather than having to pick (almost) arbitrary a time between traditional snapshots.

Unfortunately, this requires several new fields for each field written in the logfile and therefore increases considerably the memory required for the simulation. This increase will also result in larger MPI communication and, thus, a slow down of the simulation.

A simpler solution is to use a criterion based on the number of steps since the last record (i.e. write every N active steps). As the particle time-step size is based on the time scale allowing to correctly integrate the particle's properties (e.g. CFL condition), the quality of the future interpolation is hence directly linked to the quality of the actual simulation integration.

2.7. Parallelism considerations

As the CSDS relies heavily on the access to the logfile (both for the reading and writing), a careful design of the I/O and good strategies to deal with shared and distributed memory parallelism are required. Firstly, the I/O needs to ensure a low number of access to the data storage device in order to reduce the impact of the storage latency. Secondly, the CSDS needs to be protected against race conditions due to parallel processing. The solution to these challenges differs depending on the type of simulations (shared memory or distributed) and is therefore described separately.

2.7.1. Memory-mapped file

The accesses to the logfile can dramatically decrease the performances of the CSDS due to the storage latency, therefore direct access through the system functions `write` and `read` is strongly discouraged as they will perform an operation on the file storage at each call.

In order to avoid this problem, operating systems (OS) provide functions to memory-map a file and perform lazy operations on it (see e.g. `mmap` for the POSIX standard). This means that the OS will load new pages[4] only when requested. When the memory accesses are predictable, the OS can load them in advance. The pages will be kept in the memory until the OS estimates that they will not be accessed anymore. When unloading a page, the OS lazily copies it to the file storage.

It is also worth pointing the fact that this type of memory-mapped file is very easily manipulated through the use of a pointer. This pointer behaves exactly as if the whole file was loaded in memory and the OS manages everything in the background.

In case of failure or crash, all the information since the last `mmap` synchronization will be lost, but for the rest of the file, everything will be left intact. In order to ensure a safe restart of the simulation, it is possible to manually synchronize a memory-mapped file between the RAM and the file system (`msync`). This manual synchronization should be kept to a strict minimum and should be called only when writing the restart files.

2.7.2. Shared memory strategy

In shared memory parallelization, two threads can try to access and update the same bits at the same time which results in a race condition. To avoid this problem, so-called *atomic* operations have been developed in most programming languages in order to ensure that a single thread access a variable at a given time. Unfortunately, the atomic operators slow down the execution of the code and therefore should be used as little as possible.

For the CSDS, when writing a record to the logfile, a race condition can arise as two threads may decide to write at the same place and erase the work of the other. To avoid this problem, the memory-mapped file does not need to be protected, but only the pointer to the next free bytes. When a thread needs to write a record, it simply computes the record's size, and then increments the pointer by the size with an atomic operator. It can later freely write on the assigned memory. This can be done for multiple particles at a time to decrease the amount of atomic operations.

2.7.3. Distributed memory strategy

Unfortunately, the simple approach highlighted above works only for shared memory models, thus in distributed memory parallelization, a different technique is required.

Two different possibilities are often used when dealing with files. Either each MPI rank owns a single file or we can predict the required amount of memory used by each rank and can then safely write synchronously in different part of the file. Even if the memory could be easily predicted by checking all the active particles before the beginning of a time step, we decided to use a single logfile per rank for the CSDS as it avoids these additional computations.

For the index files, we follow the same approach and write a set of files for each rank. Another advantage is that with `MPI`, the simulation is usually split in sub-volumes (domains) and each volume belongs to a single rank. This means that a distributed approach allows us to not read the entire logfiles when looking at only a small volume of the simulation for analysis. All is required is a bunch of meta-data describing which sub-volume is in which

[4] A page corresponds to a part of a file.

sub-file. We note that an approach using only MPI would also be possible with one logfile per rank now corresponding to one logfile per compute core. The overall CSDS mechanism is agnostic of the underlying domain and parallelization strategy.

As the different ranks are exchanging particles (e.g. when they move out of their domain), the CSDS needs to follow the particles when they are leaving/entering a rank. In the logfile, a particle record is written when the particle leaves (enters) the rank and contains a flag giving the ID of the other rank. In the index file, the particle is simply considered as being created or deleted and thus is written in the history.

3. Implementation in the SWIFT code

SWIFT (Schaller et al., 2016, 2018) is an open source code designed for cosmological hydrodynamical simulations but also used in planetary science (Kegerreis et al., 2019) and with extension to engineering applications (Bower et al., 2021). We tried to implement the CSDS as independently as possible from the rest of SWIFT and recommend interested readers to copy/adapt the writer from SWIFT into their own code. While the writer is obviously somewhat implementation-dependent due to the differences in physics and implementation, the reader, however, should be quite universal.

3.1. Specifics of the SWIFT code

SWIFT uses MPI in-between ranks and a task-based parallelism approach relying on pthreads (Gonnet et al., 2016) within rank. This latter approach makes it an excellent candidate for the CSDS as it is able to deal with the CSDS output whilst doing some other computations. Thus it reduces the stress inflicted to the storage device due to the lower number of threads accessing the files concurrently and should provide a higher efficiency than if all the threads were writing at the same time (as would be the case with a more traditional parallelism approach).

SWIFT is designed as an HPC code that interacts the different types of particles[5] together with a minimal knowledge of the underlying interaction models. It means that the code can easily switch between different type of simulations (e.g. cosmology, planets or engineering). During the development of the CSDS, we also tried to follow this modular approach and included different writing strategy for the different particles and physics modules.

3.2. Implementation of the CSDS in SWIFT

In the first sections, an overview of the CSDS was given. As it does not include some deep technical details, we will cover the most important of them in this section. Let us start with our management of the file size followed by masking details.

As we cannot predict the final file size before running the simulation, we need a mechanism to increase the file size if the expected size is not sufficient. This is done in between every time-step where we ensure that the file is large enough to write at least all the particles once with all their fields (even if it does not happen). If it is not the case, the size of the logfile is increased by this amount times a factor in order to avoid increasing it at every step. This operation can be expensive for large file sizes, therefore the user should try to estimate accurately the size and over-allocate a bit. Future versions will attempt to make this guessing work less necessary by providing a heuristic. It is worth mentioning that at the end of the simulation, the file is truncated in order to match the exact space used by the logfile.

Due to our modular approach, the masks need to be assigned in a dynamic way as each module (gravity, SPH, etc.) defines its own fields to write. Thus, depending on the configuration options, the number of fields can vary between runs. The masks are written in the first 2 bytes of the records' header; the offsets use an additional 6 bytes. This implies we can only have $2 \times 8 = 16$ masks including the ones for the time and the special flag. Therefore we cannot log too many fields individually and have to group them into a single mask.[6] In an effort to reduce the number of masks needed, we check if two types of particles define the same mask (e.g. positions) and assign the same value to both of them if it is the case.

3.3. Special cases

During a simulation, the particles can go through different processes such as type transformation, creation, or suppression. For example, in cosmological simulations, the star formation is usually done through the transformation of a gas particle into a star particle or could be done, in a less conventional way, by directly creating a new star; the black holes destroy some particles through black hole mergers or gas absorption. For engineering usages, aerodynamics studies use wind tunnel simulations where some particles are created on one side and removed on the opposite side in order to simulate a wind. Finally, in the case of non-periodic boundaries condition (e.g. planet simulations), the particles can leave the box and therefore are removed from the simulation.

The three different cases (creation, suppression and transformation) can be dealt with an additional mask for the particles and an additional writing of the particles before and after each event. In the data, we store a single integer that includes both a flag for the type of event (creation, suppression, transformation) and the related data (e.g. the new type of the particle). For the index files, we also keep in memory the suppressed and created particle events (offset in the logfile and id of the particle) and write them in the next index file.

3.4. MPI strategy

In SWIFT, the volume is split in smaller volumes that are distributed to the different ranks according to some evaluation of the work required for each sub-volume. Throughout a simulation, the volumes tend to stay attributed to the same MPI rank in order to avoid too much communications, therefore a particle leaving (entering) a given volume can be considered as being removed from (created in) a given rank without having a large impact on the logfile size. The only difference with the suppression and deletion presented in the previous section (i.e. due to physics events) is that we use a different flag and store the id of the other rank involved in the transaction. We also ensure that the offsets are correctly written between the two files.

4. Reading strategies

The format of the logfile and index files is not trivial and cannot be easily read. Therefore a library is required to read the CSDS and makes it more accessible to the users. This library (or reader) is implemented in `C++` alongside a `python` wrapper. A documentation is provided in the repository. We also provide a

[5] In cosmological simulations, we use up to 7 different types of particles representing the gas, dark matter (2 types), stars, black holes, sink particles and neutrinos.

[6] Depending on the future usage of the CSDS, we might need to change the masks from a field point of view to a particle type point of view (e.g. one mask per particle type and not per field).

few functions in C in order to write the files. As was the case for the writer, the core implementation of the reader is fully independent of the physics model. Only the reading and interpolation functions need to be updated when adding new fields.

The reader naturally provides two important possibilities: (1) follow a particle (or a set of particles) through time in an accurate and efficient way, and (2) generate a traditional snapshot-like output from the CSDS. While the generation of snapshots is not the best usage of the CSDS, it can greatly improve the compatibility of the CSDS with existing analysis codes and provides a speedup when the user needs to do a deep analyze of a single specific time (e.g. at redshift $z = 0$).

In order to speedup the reading process, the reader does not read directly the logfile (except when reversing the offset of the records during the first reading), it always starts by finding the particles in the index files and then read the logfile starting from the last offset of the particles of interest.

When reading a subset of the particles, it is important to have an efficient way of finding the offset of the particles. This is done by sorting the index files according to the IDs of the particles and then using a bisection search. As the initial order of the particles does not represent anything meaningful, the index files can be saved once sorted.

In the case of simulations done with multiple `MPI` ranks, the reader will independently read each logfile and the corresponding index files. It means that the user needs to ensure that all the logfiles are read and, then, concatenates the output together if needed. A more thorough parallelization of this process is left for future work.

4.1. Implementation details

In this section, the different steps done by the reader are summarized in order to give an overview of the method. We focused on the most important pieces.

The reader starts by reading the header of the logfile and checks if the offsets are already in the correct direction. If it is not yet the case, the reversal operation is done immediately. This is done by starting with the first record in the logfile. If a previous record exists, its offset is modified in order to point to the current record. Once that record has been processed, the reader moves to the end of the current record and starts over with the next record. This requires a single linear reading of the file with random accesses to some previous part of the file. It is worth mentioning that, in its current implementation, if this process is interrupted, the file will be corrupted and it will be complicated to restore it.

Once the offsets have been reversed, the reader reads all the time records and populate a lookup table with them in order to have a quick access to the time of the particle records. As this operation can be long depending on the file size,[7] the array is saved at the end of the first index file and can be restored in future readings. The reader finalizes its initialization by reading all the index files headers and storing their time. This concludes the first read and post-processing of the log files.

When requesting some data, the reader starts by setting the current time. This operation is done by selecting the two correct index files: the first one gives the information about the last known state of the simulation and the second one provides the information about the particles created and removed. Then the number of available particles is computed from the index files for the allocation of the output arrays. The particles are read one after the other assuming that they are still present at the required time. If it is not the case, the code simply skips the current particle and goes to the next one. As this approach could raise some issues if the index files or the logfile are corrupted, we perform a sanity check at the end in order to verify that we have the correct number of particles and raise an error if it is not the case. It would be possible to predict which particles are still present at a given time thanks to the index files, but this would require too much work for the particles that are still present. Note that, as we do not expect a lot of particles to be removed from the simulation, we decided to use the approach described rather than the prediction.

The particles are read field by field in order to be more flexible when each field is written at a different frequency. As the file is memory mapped, we always read the same file pages and they should stay within the memory during the whole reading of a single particle. Thus, even if we read more often the file and do a bit more operations, the overall efficiency of the code stays the same.

Once all the previous operations are done, the output arrays are given back to the user. As we are reading the particles in the same order than the index files, the output arrays are sorted in the same way (e.g. by particle type and then ids).

5. Results

As mentioned in the introduction, our aim is to provide a way to recover the state of the simulation at any time with a high accuracy and a low storage cost. To demonstrate this, we used an isolated disk galaxy ran as a dark matter only simulation containing 360'000 dark matter particles within an Hernquist potential.[8]

Using a snapshot as our reference point, Fig. 3 shows the relation between the reconstruction accuracy of the particles' orbit within the halo and the storage required for both the CSDS and the traditional snapshot strategy. In this graph, both the snapshots and the CSDS use a cubic Hermite interpolation based on the positions and velocities (see Appendix A) to reconstruct data at intermediate times. The accuracy is measured by the relative error on the orbit position (here the radius). The relation for the snapshots is obtain through an interpolation of the reference snapshot from two snapshots written at an equal distance in time from the reference. The CSDS decreases the error by a factor of 67.8x at a given storage cost or decreases the storage cost by a factor of 2.76x for a given error.

Let us now look at the convergence of the CSDS. In this case, the interpolation of the position is done through a quintic Hermite interpolation. Due to its high convergence order, it does not require many points to accurately represent the orbit of a particle. In order to illustrate this point with the CSDS, in Fig. 4 we show a simulation of a planet in orbit around a star. The interpolation is done with the help of the velocity and acceleration to constrain the two first derivatives in the interpolation. While the CSDS is set at low time resolution, the traditional snapshots were written at high resolution to allow for a fair comparison. During a single orbit, only three records (in orange) were written. As expected, the positions of the orange crosses perfectly match the positions of the snapshots (in blue). The interpolation given in red slightly differs with the snapshots far away from the records, but provides overall a correct interpolation. For cosmological simulations, the situation is more complicated than in this example, but we can already see here that a low number of points are sufficient to properly reconstruct the orbits.

[7] Comparable time to reading the entire state of the simulation (3.0s for the time array and 4.6s for the state in the millennium simulation with $N = 384^3$, $\Delta n = 10$ shown in Section 5).

[8] The initial conditions and parameters are fully provided within Swift's repository and correspond to the `IsolatedGalaxy_dmparticles` example.

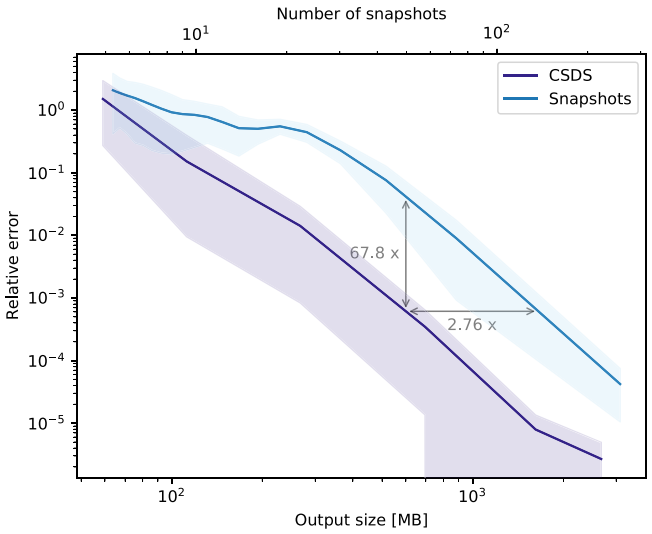

Fig. 3. Relative error of the radial position of the particles as function of the output size for both the traditional snapshots and the CSDS. Both relations are computed from the same simulation that consists of an isolated disk galaxy simulated only with gravity. Only particles with $r \in [5, 10]$ kpc are taken into account in this figure. The average error is shown as the straight lines and the distribution is shown with the areas representing the 16/84 percentiles. For both the CSDS and the snapshots, a cubic interpolation is done. The CSDS requires less data storage for a fixed reconstruction accuracy. Or, equivalently, the CSDS leads to a better reconstruction for the same storage.

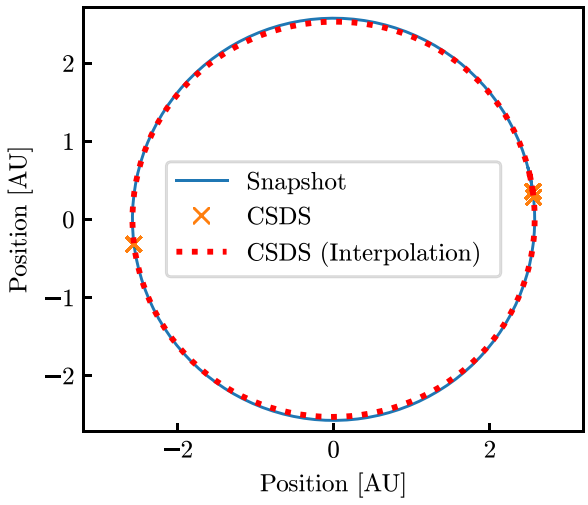

Fig. 4. Orbit of a planet around a star reconstructed with the CSDS (red line and orange crosses) and the snapshots (blue line). The orange crosses correspond to the place where the CSDS writes a record. The quintic Hermite interpolation (using the velocity and acceleration) allows to accurately reconstruct the orbit with a small number of points.

Using the isolated galaxy simulation previously described, in Fig. 5 we show the relative error of the CSDS as function of the number of steps between writing (Δn) for different bins of distances from the galaxy center. As one can expect, the CSDS converges towards the correct solution when increasing the output frequency (meaning decreasing Δn). This relation can be used by users to set a reasonable value of Δn for their own simulations.

Finally, we demonstrate the use of the CSDS on a real production simulation. In Fig. 6 we show the scaling in terms of output size and reading speed as function of the time resolution of the CSDS (Δn) and the number of particles for a cosmological volume. More precisely, this is the `PMillennium-384` example

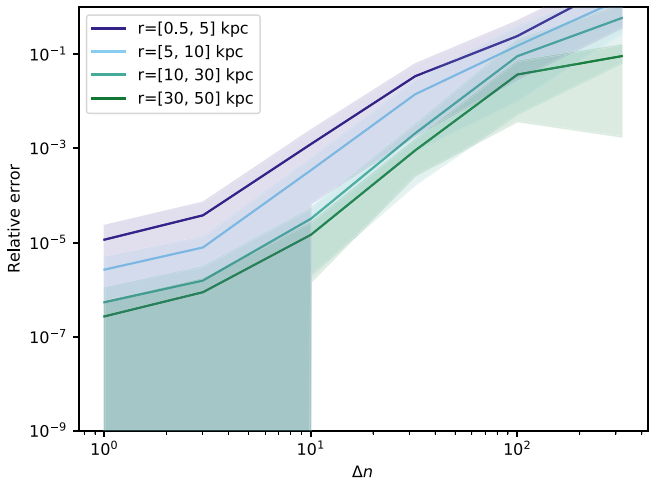

Fig. 5. Relative error of the radial positions of the particles as function of the number of steps between two dumping of records for an isolated disk galaxy ran only with gravity. The interpolation is done through a quintic Hermite interpolation using the positions, velocities, and accelerations stored in each record. The means of the distributions are shown with the solid lines and their 16/84 percentiles are indicated using shaded areas. As expected, the CSDS converges towards the solution provided by a snapshot.

provided within the Swift code. It contains between $N = 384^3$ and $N = 1536^3$ particles within a volume of 800^3 Mpc^3; i.e. a low-resolution version of the P-Millennium simulation of Baugh et al. (2019). The number of particles within the first graph is 384^3 and the CSDS time resolution in the bottom panel is $\Delta n = 100$ steps. The largest simulation ran is within the same volume but with $N = 1536^3$ particles and with exactly the same physics. As the reading speed is dependent on the index files, two different lines are shown. The best case scenario (in red) corresponds to the case where the requested time matches an index file ($z \approx 0.12$). The worst case scenario (in orange) corresponds to the case where the requested time is just below an index file ($z \approx 0$). It means that the index file used is the same as in the best case scenario, but the particles need to be updated over 1.6 Gyr. For the `MPI` scaling, we have seen some relatively modest changes in terms of performances. On the $N = 384^3$, moving from 1 rank to 16 ranks produced a storage increase of 5% and a variation of reading speed within the error margin (1%).

We note that the simulations here are relatively small ($\lesssim 10^9$ particles) when compared to state-of-the-art simulations (e.g. Bocquet et al., 2016; Ishiyama et al., 2021), however, the data ratios between the CSDS and snapshots approaches remain constant when increasing the simulation volume. We hence expect the approach to carry over and increase linearly with the number of particles.

6. Discussion

While the snapshots and the CSDS are writing the same information, the format is far too different (single time vs whole simulation) to have a fair and complete comparison. To make it even worse, they will not behave in the same way depending on the type of simulations (e.g. only hydrodynamics vs with gravity) due to the differences in the hierarchy of time-step sizes and variability of the fields to write. The gravity produces strong differences in timescales and thus will improve the CSDS performances when compared to hydrodynamics simulations without any gravity that can be easily described by a single time step. It means that the CSDS performances will strongly depend on the type of simulation. As the CSDS was developed for simulations

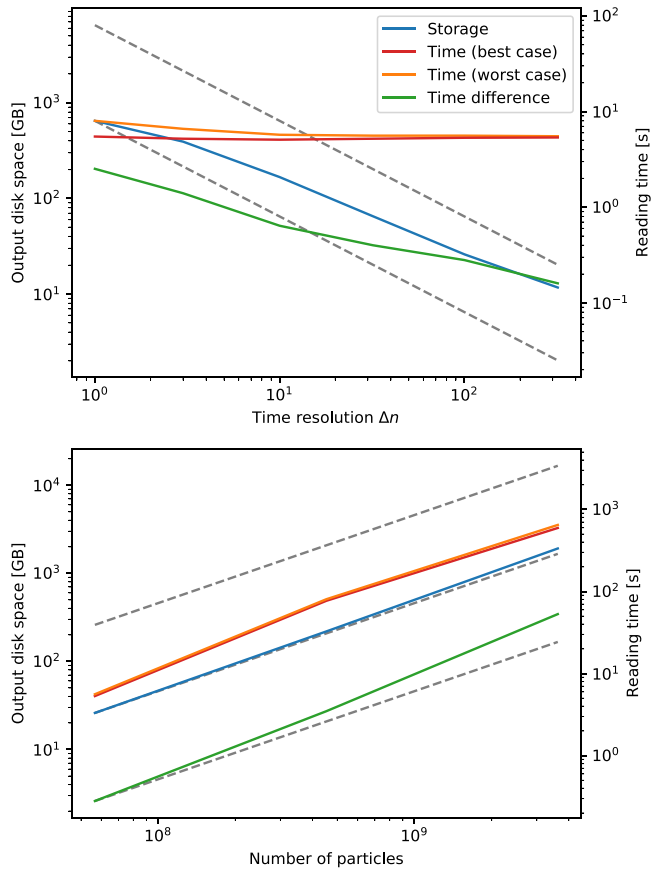

Fig. 6. Scaling of the CSDS both in terms of output size and reading time for a small version of the simulation presented by Baugh et al. (2019). For the first graph, the number of particles is 384^3 and, for the second graph, $\Delta n = 100$. The first graph shows them as function of the time resolution of the output (number of steps between records Δn). The second graph shows them as function of the simulation size. The dashed black line corresponds to a linear scaling with a factor of 10 between them. The best case scenario is a requested time that corresponds to an index file. The worst case scenario is a requested time that is just before an index file. In green, the difference between the best case and the worst case is shown.

including gravity, we mainly focused on such simulations in this paper.

In Fig. 3, we tried to show the relation between the accuracy and the storage cost for both the CSDS and the snapshots on a simulation that includes gravity but no hydrodynamics. As mentioned in the previous section, the error of the snapshots is obtained through an interpolation. We picked a reference snapshot at equal distance from the snapshots used for the interpolation. It means that the snapshot error represents the *worst* case scenario. With the CSDS, there is no global worst case scenario, only individual ones that are represented with the upper limit of the distribution. The CSDS clearly outperforms the snapshots in term of accuracy (67.8x) but only moderately decreases the output storage which was our initial aim. We note that more advanced interpolation techniques, for instance for particles in haloes (e.g. Smith et al., 2022) could also be used to improve the accuracy, or reduce the number of entries needed to reach the same precision.

In Fig. 5, we showed a convergence study of the relative error with respect to the number of steps between writing. As it can be seen, at any point on the orbit, the CSDS converges towards the correct solution. This graph shows that the CSDS is able to recover the orbits of all the particles including the most chaotic ones close to the center of a halo. We can see that writing every 10 (gravitational) time-steps is sufficient to produce positions with an accuracy of 0.1% even close to the galaxy's center.

Finally, we looked at the scaling of the output size and reading speed as function of both the output frequency (Δn) and the number of particles in the simulation. As expected, they both depend linearly on the number of particles. In the case of the storage cost, the scaling is also more or less linear as function of the time resolution, but that will depend on the exact time-step hierarchy distribution.

Let us now focus on the reading time as function of Δn. The best case scenario is when a user is reading an index file and then directly reading a value from the logfile without interpolation (i.e. the data requested corresponds directly to one of the time records). It means that the time resolution will have no impact on it and thus the reading time is independent of Δn. The worst case scenario is more complex as it depends on when the index files are written and on the hierarchy of time-step sizes. In cosmological simulations, most of the particles will have large time steps (see Fig. 1). Therefore, the average number of "jumps" in the logfile required to update the particles will be relatively low (between 1.0 and 7.2 for our different time resolutions in this example). The extra time required to evolve the particles (i.e. finding the correct records and interpolating) is almost negligible in all our simulations shown here even if the evolution interval is relatively large (1.6 Gyr). This demonstrates the high efficiency of the evolution of the records in comparison to a single reading of the simulation.

In terms of MPI scaling, the increase of storage is expected as all the particle exchanges will be written into the logfile. Thus the results will strongly depends on the MPI splitting strategy. In this case, at the beginning, we distribute the particles in order to have the same number of particles on each rank. Then the corresponding volume is assigned to a rank and it keeps it until the end of the simulation. The almost constant reading time can be explained by the fact that the storage spreads over multiple files which increase the chance of cache hit and thus reduce the time lost in file accesses.

When comparing with traditional snapshots (e.g. 13.4s for a simulation with 768^3 particles), the reading time of the CSDS is larger but still manageable. While this can be an issue, it is worth to explicitly mention that the main advantages of the CSDS are the accurate reconstruction at any time *and* the possibility to evolve the particles quickly once obtained (see the difference between the best and worst cases). In any case, the CSDS mechanism is still at an early development state and will certainly benefit from optimizations in the coming years. In the meantime, if high performance is required (or if analysis tools require data laid out in traditional way), it is always possible to construct a snapshot from the CSDS reader and then work with it.

Some possible optimizations for future versions include tools to help with the analysis of a completed simulation. It is often useful to be able to select the particles in a sub-region (e.g. in a given halo). In a snapshot-based strategy this is often achieved by sorting the particles at the end of a simulation based on some coarse-grained grid or space-filling curve. The same approach could be applied to the index file in the CSDS case. This would then let users rapidly retrieve particles in regions of interest. Similarly, tools developed for public release of data via web portals (e.g. Ragagnin et al., 2017) could also be modified to retrieve the particles of interest at any point in time based on this index file sorting and the interpolation mechanism of the CSDS.

7. Examples of application

In this section we showcase some application examples that directly benefit from the CSDS output strategy.

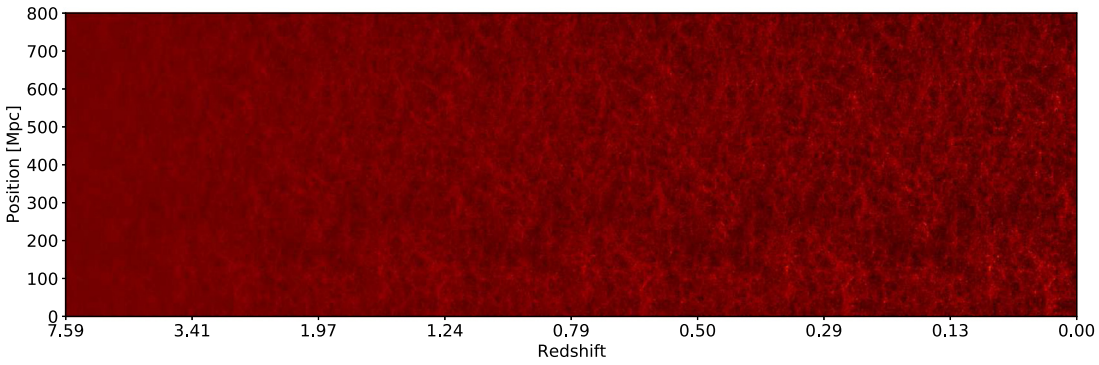

Fig. 7. Light cone image with the time axis along x. The image consists in a single volume where the time of each particle is selected in order to emit lights such as it reaches the right side at z = 0. With the snapshots, it is required to read the particles by slices and thus produce discontinuities in the image. Thanks to the CSDS, this is no longer the case as we can find with high accuracy the time when a particle is crossing the light cone.

A problem in the comparison of large scale simulations with observations comes from light travel times. As in observations, the further the observed objects are, the older we observe them to be. We need to produce the same behavior in our simulations. This is typically done with the method called light-cones (e.g. Evrard et al., 2002; Garaldi et al., 2020) and the main idea behind it is to extract slices of volume in each snapshot and to stack them together before projecting along time. Due to the snapshots, this can be only done approximately or with an extremely large number of snapshots. Garaldi et al. (2020) solved this issue by implementing the light cone computation directly into the simulation code and redesigned it in order to reduce the resolution in the area outside the light-cone. Thus, a single light-cone is selected at the beginning of the simulation and cannot be changed. With the CSDS, it is possible to generate them after the end of the simulation and for as many light cones as desired thanks to the high time resolution in the CSDS along with the interpolation. Instead of constructing snapshots as a slice in time, we can construct slices in space–time alongside a constant light geodesic. For example in Fig. 7, a light cone image is produced from a normal CSDS output, where the position of the observer and light-cone properties were chosen *after* the completion of the run. On the figures, the axis was switched (we project both the time and position along the x axis) in order to show the time evolution.

Another usage of the CSDS is the study of fast events. To illustrate this, in Fig. 8, a phase diagram is shown for a cosmological simulation of a dwarf galaxies, here using the (Revaz and Jablonka, 2018) model. The diagram consists of a histogram of the distribution of the density and internal energy of the gas at redshift 6 and the time evolution of a reference gas particle between $z \in [23, 4]$ (yellow to purple). The gas inside the galaxy is in a dense phase (above 10^{-2} atom/cm^3) and can be split into the cold (below 10^{10} erg/g) and hot phase that has been recently touched by a supernova. Inside a galaxy, the behavior of the gas is extremely chaotic due to the constant explosion of various supernovae. In these simulations, supernovae are modeled by directly injecting some energy into the surrounding gas particles and provoke a quick vertical displacement in the phase diagram. While the timescale of the impact of a single supernovae is typically less than a few Myr, the cosmological simulations are usually done over 14 Gyr. The two different timescales make it very hard to accurately follow the impact of supernovae with the traditional snapshots. The CSDS is perfectly able to resolve both timescales at the same time and can help to enhance our understanding of supernovae in cosmological simulations. We note, however, that more advanced choices for the frequency of dumps in the log might be necessary to fully capture all the events, especially the ones around shocks or other discontinuity

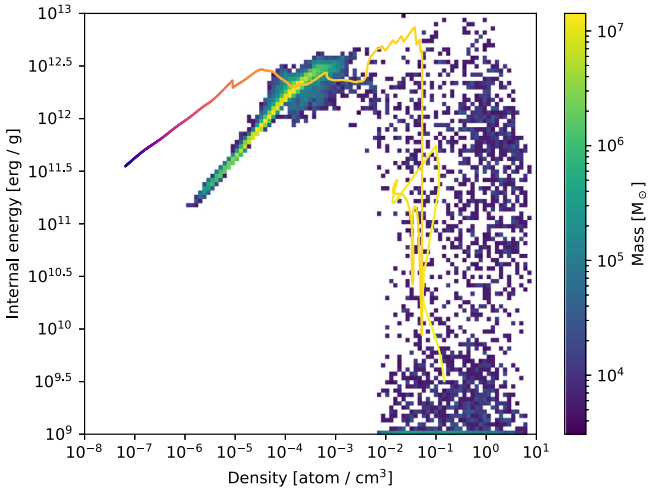

Fig. 8. Phase diagram of the density and internal energy in physical units for a simulation of the dwarf galaxy 159 with GEAR's model (Revaz and Jablonka, 2018) in SWIFT at redshift 6 for the background. The line shows the evolution of a particles between redshift 23 (yellow) to 4 (purple). With the CSDS, we can accurately follow the evolution of the particle in the chaotic and dense medium and clearly see when a supernovae directly touch the particle (straight vertical line). Due to the evolution between redshift 6 and 4, the final position of the particle followed does not match the rest of the low density particles.

in the fluid. This could, for instance, include the forceful addition of an entry in a logfile when a shock is detected in between regularly scheduled entries. We leave the exploration of such improvements to future work.

Finally, our last example of usage is in the production of movies. The CSDS allows for a higher time resolution than when using traditional snapshots and is therefore particularly useful in this case as the producer is not limited by the available snapshots. Thus the movie's speed can be easily adapted around specific times or events. Three movies have been produced for this paper and are hosted on YouTube (described in detail on YouTube): a cosmological simulation[9] (https://www.youtube.com/watch?v=OKKsk0TigNo), the chemical evolution of the same simulation (https://www.youtube.com/watch?v=5AqmAUGndps), and a planetary impact (https://www.youtube.com/watch?v=6aBry0CUgVw). While the first and last movies show the high time resolution of the output system for a cosmological simulation and a planetary impact, the second one shows that it can be used directly for scientific analysis of the runs.

[9] The original movie without YouTube's compression is available in the movie's description. This simulation is the same as shown in Fig. 8.

8. Conclusions

We presented a new output technique that can follow the evolution of all the resolution elements of a simulation and write them using their own individual timescale in order to reduce the output size and increase the accuracy at any time. This approach is particularly well fitted for simulations that include gravity as it produces large differences in time scales between resolution elements. On the example presented in Fig. 3, we increased the accuracy of the output by 67.8x for a given storage space (or a factor of 2.76x in storage space for a given accuracy). This technique largely improves on the traditional snapshot method and will allow to improve the analysis of fast events (e.g. supernovae) within simulations. While this technique has been specially created for particle-based discretization, it could be readily adapted for grid-based techniques.

Finally, it is worth highlighting that this is a first presentation of the method and that many improvements will be brought in the future. Among them, it is worth to mention a few ideas that we would like to explore. Currently, the index files do not contain any physical information. It could be possible to sort the particles according to a tree in order to speedup any spatial selection. In most astrophysical codes, such a tree is already present in the simulation and it would hence not require a lot of computation to produce such tree-sorted files. Another improvement could be done on complexifying the frequency writing criterion. In large scale simulations such as EAGLE, writing the whole simulation at high resolution requires an extremely large storage. To reduce it, it would be possible to adapt the writing frequency according to some physical properties (e.g. write if a given property has changed by more than 10%) or spatial position (e.g. focus on a single galaxy). This could help to achieve an extremely high time resolution within a single galaxy. A last interesting improvement in terms of physics could be to increase the interpolation order by the usage of more than 2 records which would decrease even further the storage cost for a given accuracy.

CRediT authorship contribution statement

L. Hausammann: Methodology, Software, Investigation, Writing – original draft, Visualization. **P. Gonnet:** Conceptualization. **M. Schaller:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

We thank the anonymous referees for raising interesting points which improved the quality of this paper. We gratefully acknowledge the help of Bert Vandenbroucke and Alexei Borissov to test the CSDS, as well as Jacob Kegerreis for help running the planetary simulation test-case. We are also grateful for the ideas raised by Mladen Ivkovic, Yves Revaz and Richard Bower during our discussions. We thank Alyson Brooks for reviewing an early version of this paper. MS is supported by the Netherlands Organization for Scientific Research (NWO) through VENI grant 639.041.749. This work was supported by the Swiss Federal Institute of Technology in Lausanne (EPFL) through the use of the facilities of its Scientific IT and Application Support Center (SCITAS) and the University of Geneva, Switzerland through the usage of Yggdrasil. This work used the DiRAC@Durham facility managed by the Institute for Computational Cosmology on behalf of the STFC DiRAC HPC Facility (www.dirac.ac.uk). The equipment was funded by BEIS capital funding via STFC capital, United Kingdom grants ST/K00042X/1, ST/P002293/1, ST/R002371/1 and ST/S002502/1, Durham University and STFC operations, United Kingdom grant ST/R000832/1. DiRAC is part of the National e-Infrastructure. We are grateful to the Numpy (Oliphant, 2015), Matplotlib (Caswell et al., 2018) SciPy (Jones et al., 2001) and IPython (Perez and Granger, 2007) teams for providing the scientific community with essential python tools. The research in this paper made use of the SWIFT open-source simulation code (http://www.swiftsim.com, Schaller et al., 2018) version 0.9.0.

Appendix A. Interpolation with Co-moving coordinates

The Hermite interpolation between time t_0 and t_1 at t is given by:

$$p_n(u) = \sum_{i=0}^{(n-1)/2} p_0^{(i)} H_i^n(u) + p_1^{(i)} H_{n-i}^n(u) \tag{A.1}$$

where n is the degree of the interpolation, $p_0^{(i)}$ ($p_1^{(i)}$) is the ith derivative at t_0 (t_1) and u is the normalized time variable ($(t - t_0)/(t_1 - t_0)$). The polynomials H_i^n are given by the line of the following matrix where the first column corresponds to the constant term (Lind, 2020):

$$H^3 = \begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 3 & -2 \end{pmatrix}. \tag{A.2}$$

For example $H_0^3(u)$ is given by $1 - 3u^2 + 2u^3$. For the quintic interpolation, the matrix is given by:

$$H^5 = \begin{pmatrix} 1 & 0 & 0 & -10 & 15 & -6 \\ 0 & 1 & 0 & -6 & 8 & -3 \\ 0 & 0 & \frac{1}{2} & -\frac{3}{2} & \frac{3}{2} & -\frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & -1 & \frac{1}{2} \\ 0 & 0 & 0 & -4 & 7 & -3 \\ 0 & 0 & 0 & 10 & -15 & 6 \end{pmatrix}. \tag{A.3}$$

In cosmological simulation, co-moving coordinates are required to take into account the expansion of the universe (for more information on co-moving coordinates see for example Bertschinger, 1998; Peebles, 1993). In SWIFT, they are defined as $x = ax'$, $a^2\dot{x}' = v'$ and $a\dot{v}' = g'$[10] where x', v' and g' are the co-moving coordinates, velocities and accelerations, and a is the scaling factor that depends on time.[11] To use the Hermite interpolation previously described, u is replaced by $(a - a_0)/(a_1 - a_0)$ and the terms $p_0^{(i)}$ and $p_1^{(i)}$ need to include some cosmological factors. Those last terms are obtained from the derivation of the co-moving coordinate with respect to the scale factor. For the positions, the velocities $p_0^{(1)}$ and $p_1^{(1)}$ need to be modified with the following expression evaluated at either a_0 or a_1:

$$\frac{\mathrm{d}x'}{\mathrm{d}a} = \frac{\mathrm{d}x'}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}a} = \frac{v'}{a^2\dot{a}}. \tag{A.4}$$

The derivative of a with respect to time is given by:

$$\dot{a} = aH_0\sqrt{(\Omega_c + \Omega_b)a^{-3} + \Omega_{rad}a^{-4} + \Omega_\Lambda} \tag{A.5}$$

[10] The last one holds only for gravity as hydrodynamics have a different relation.

[11] Size of the universe normalized to today's value.

Listing 1: Example of the CSDS's interface.

```
import numpy as np
import libcsds as csds

# Basename of the logfile
BASENAME = "index_0000"
# The ids of the particles that we wish to follow
PARTICLE_IDS = np.array([10, 412, 213], dtype=int)
# The simulation time
TIME = 0.5


# Open the CSDS
with csds.Reader(BASENAME, verbose=0) as reader:
    # Get the time limits within the logfile
    t_min, t_max = reader.get_time_limits()

    # Ensure that the time is correct
    if TIME < t_min or TIME > t_max:
        raise Exception("The requested time is unavailable.")

    # Ensure that the fields are available for the particle type
    fields = reader.get_list_fields(part_type=csds.gas)

    if "Coordinates" not in fields or "Entropies" not in fields:
        raise Exception("Field not found in the logfile")

    # Read a subset of particles (coordinates and entropies are
    # 2 numpy arrays)
    coordinates, entropies = reader.get_data(
        fields=["Coordinates", "Entropies"], time=TIME,
        filter_by_ids=PARTICLE_IDS)

    # Read all the gas particles
    all_fields = reader.get_data(
        fields=fields, time=TIME, part_type=csds.gas)
```

where Ω are the cosmological parameters and H_0 the Hubble constant. This equation assumes a flat Λ-CDM universe. For the accelerations within the position interpolation ($p_0^{(2)}$ and $p_1^{(2)}$), an additional derivative is required:

$$\frac{\mathrm{d}^2 x'}{\mathrm{d}a^2} = \frac{1}{a^4\dot{a}^2}\left(ag' - 2a\dot{a}v' - \frac{a^2\ddot{a}}{\dot{a}}v'\right). \tag{A.6}$$

In the case of the interpolation of velocities, the same computation is required for the accelerations ($p_0^{(1)}$ and $p_1^{(1)}$) and gives:

$$\frac{\mathrm{d}v'}{\mathrm{d}a} = \frac{g'}{\dot{a}a} \tag{A.7}$$

Appendix B. Example of python scripts

The following code shows a quick example of the CSDS's interface. The full interface is described in the documentation provided within the reader's repository.

References

Aarseth, S.J., 1963. Dynamical evolution of clusters of galaxies, I. Mon. Not. R. Astron. Soc. (ISSN: 0035-8711) 126, 223. doi:10.1093/mnras/126.3.223, URL http://adsabs.harvard.edu/abs/1963MNRAS.126..223A.

Abbasi, H., Lofstead, J., Zheng, F., Schwan, K., Wolf, M., Klasky, S., 2009. Extending I/O through high performance data services. In: 2009 IEEE International Conference on Cluster Computing and Workshops. pp. 1–10. doi:10.1109/CLUSTR.2009.5289167, ISSN: 2168-9253.

Adams, M.F., Ku, S.-H., Worley, P., D'Azevedo, E., Cummings, J.C., Chang, C.-S., 2009. Scaling to 150k cores: Recent algorithm and performance engineering developments enabling XGC1 to run at scale. J. Phys. Conf. Ser. (ISSN: 1742-6596) 180 (1), 012036. doi:10.1088/1742-6596/180/1/012036, URL http://iopscience.iop.org/1742-6596/180/1/012036.

Baugh, C.M., Gonzalez-Perez, V., Lagos, C.d.P., Lacey, C.G., Helly, J.C., Jenkins, A., Frenk, C.S., Benson, A.J., Bower, R.G., Cole, S., 2019. Galaxy formation in the Planck Millennium: the atomic hydrogen content of dark matter halos. Mon. Not. R. Astron. Soc. 483 (4), 4922–4937. doi:10.1093/mnras/sty3427, arXiv:1808.08276.

Bertschinger, E., 1998. Simulations of structure formation in the universe. Annu. Rev. Astron. Astrophys. (ISSN: 0066-4146) 36, 599–654. doi:10.1146/annurev.astro.36.1.599, URL http://adsabs.harvard.edu/abs/1998ARA%26A..36..599B.

Bocquet, S., Saro, A., Dolag, K., Mohr, J.J., 2016. Halo mass function: baryon impact, fitting formulae, and implications for cluster cosmology. Mon. Not. R. Astron. Soc. 456 (3), 2361–2373. doi:10.1093/mnras/stv2657, arXiv:1502.07357.

Bower, R., Rogers, B.D., Schaller, M., 2021. Massively parallel particle hydrodynamics at exa-scale. Comput. Sci. Eng. 1. doi:10.1109/MCSE.2021.3134604.

Caswell, T.A., Droettboom, M., Hunter, J., Firing, E., Lee, A., Stansby, D., de Andrade, E.S., Nielsen, J.H., Klymak, J., Varoquaux, N., Root, B., Elson, P., Dale, D., May, R., Lee, J.-J., Seppänen, J.K., Hoffmann, T., McDougall, D., Straw, A., Hobson, P., cgohlke, Yu, T.S., Ma, E., Vincent, A.F., Silvester, S., Moad, C., Katins, J., Kniazev, N., Ariza, F., Würtz, P., 2018. Matplotlib/matplotlib v3.0.2. doi:10.5281/zenodo.1482099, URL https://zenodo.org/record/1482099#.XBDPecYo_0o.

Evrard, A.E., MacFarland, T.J., Couchman, H.M.P., Colberg, J.M., Yoshida, N., White, S.D.M., Jenkins, A., Frenk, C.S., Pearce, F.R., Peacock, J.A., Thomas, P.A., 2002. Galaxy clusters in hubble volume simulations: Cosmological constraints from sky survey populations. Astrophys. J. (ISSN: 0004-637X) 573, 7–36. doi:10.1086/340551, URL http://adsabs.harvard.edu/abs/2002ApJ...573...7E.

Garaldi, E., Nori, M., Baldi, M., 2020. Dynamic zoom simulations: A fast, adaptive algorithm for simulating light-cones. Mon. Not. R. Astron. Soc. (ISSN: 0035-8711) 499, 2685–2700. doi:10.1093/mnras/staa2064, URL http://adsabs.harvard.edu/abs/2020MNRAS.499.2685G.

Godoy, W.F., Podhorszki, N., Wang, R., Atkins, C., Eisenhauer, G., Gu, J., Davis, P., Choi, J., Germaschewski, K., Huck, K., Huebl, A., Kim, M., Kress, J., Kurc, T., Liu, Q., Logan, J., Mehta, K., Ostrouchov, G., Parashar, M., Poeschel, F., Pugmire, D., Suchyta, E., Takahashi, K., Thompson, N., Tsutsumi, S., Wan, L., Wolf, M., Wu, K., Klasky, S., 2020. ADIOS 2: The adaptable input output system. a framework for high-performance data management. 12, 100561. doi:10.1016/j.softx.2020.100561, URL http://adsabs.harvard.edu/abs/2020SoftX..1200561G,

Gonnet, P., Chalk, A.B.G., Schaller, M., 2016. QuickSched: Task-based parallelism with dependencies and conflicts. doi:10.48550/arXiv.1601.05384, arXiv:1601.05384 [Cs].

Ishiyama, T., Prada, F., Klypin, A.A., Sinha, M., Metcalf, R.B., Jullo, E., Altieri, B., Cora, S.A., Croton, D., de la Torre, S., Millán-Calero, D.E., Oogi, T., Ruedas, J., Vega-Martínez, C.A., 2021. The uchuu simulations: Data release 1 and dark matter halo concentrations. Mon. Not. R. Astron. Soc. 506 (3), 4210–4231. doi:10.1093/mnras/stab1755, arXiv:2007.14720.

Jetley, P., Gioachin, F., Mendes, C., Kale, L.V., Quinn, T., 2008. Massively Parallel Cosmological Simulations with ChaNGa. IEEE Computer Society, ISBN: 978-1-4244-1693-6, pp. 1–12. doi:10.1109/IPDPS.2008.4536319, URL https://www.computer.org/csdl/proceedings-article/ipdps/2008/04536319/12OmNvStcyx.

Jones, E., Oliphant, T., Peterson, P., et al., 2001. SciPy: Open Source Scientific Tools for Python. URL http://www.scipy.org/.

Kegerreis, J.A., Eke, V.R., Gonnet, P., Korycansky, D.G., Massey, R.J., Schaller, M., Teodoro, L.F.A., 2019. Planetary giant impacts: convergence of high-resolution simulations using efficient spherical initial conditions and SWIFT. Mon. Not. R. Astron. Soc. (ISSN: 0035-8711) 487, 5029–5040. doi:10.1093/mnras/stz1606, URL http://adsabs.harvard.edu/abs/2019MNRAS.487.5029K.

Lind, M., 2020. Real-time quintic Hermite interpolation for robot trajectory execution. PeerJ Comput. Sci. (ISSN: 2376-5992) 6, e304. doi:10.7717/peerj-cs.304, URL https://peerj.com/articles/cs-304 Publisher: PeerJ Inc..

Lüttgau, J., Snyder, S., Carns, P., Wozniak, J.M., Kunkel, J., Ludwig, T., 2018. Toward understanding I/O behavior in HPC workflows. In: 2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage Data Intensive Scalable Computing Systems (PDSW-DISCS). pp. 64–75. doi:10.1109/PDSW-DISCS.2018.00012.

Ma, X., Lee, J., Winslett, M., 2006. High-level buffering for hiding periodic output cost in scientific simulations. IEEE Trans. Parallel Distrib. Syst. (ISSN: 1558-2183) 17 (3), 193–204. doi:10.1109/TPDS.2006.36, Conference Name: IEEE Transactions on Parallel and Distributed Systems.

Maksimova, N.A., Garrison, L.H., Eisenstein, D.J., Hadzhiyska, B., Bose, S., Satterthwaite, T.P., 2021. ABACUSSUMMIT: a massive set of high-accuracy, high-resolution N-body simulations. Mon. Not. R. Astron. Soc. 508 (3), 4017–4037. doi:10.1093/mnras/stab2484, arXiv:2110.11398.

Mitra, S., Sinha, R.R., Winslett, M., Jiao, X., 2005. An efficient, nonintrusive, log-based I/O mechanism for scientific simulations on clusters. In: 2005 IEEE International Conference on Cluster Computing. pp. 1–10. doi:10.1109/CLUSTR.2005.347041, ISSN: 2168-9253.

Müller, A., Deconinck, W., Kühnlein, C., Mengaldo, G., Lange, M., Wedi, N., Bauer, P., Smolarkiewicz, P.K., Diamantakis, M., Lock, S.-J., Hamrud, M., Saarinen, S., Mozdzynski, G., Thiemert, D., Glinton, M., Bénard, P., Voitus, F., Colavolpe, C., Marguinaud, P., Zheng, Y., Van Bever, J., Degrauwe, D., Smet, G., Termonia, P., Nielsen, K.P., Sass, B.H., Poulsen, J.W., Berg, P., Osuna, C., Fuhrer, O., Clement, V., Baldauf, M., Gillard, M., Szmelter, J., O'Brien, E., McKinstry, A., Robinson, O., Shukla, P., Lysaght, M., Kulczewski, M., Ciznicki, M., Piatek, W., Ciesielski, S., Błażewicz, M., Kurowski, K., Procyk, M., Spychala, P., Bosak, B., Piotrowski, Z.P., Wyszogrodzki, A., Raffin, E., Mazauric, C., Guibert, D., Douriez, L., Vigouroux, X., Gray, A., Messmer, P., Macfaden, A.J., New, N., 2019. The ESCAPE project: Energy-efficient scalable algorithms for weather prediction at exascale. Geosci. Model Dev. 12, 4425–4441. doi:10.5194/gmd-12-4425-2019, URL http://adsabs.harvard.edu/abs/2019GMD....12.4425M.

Nelson, D., Pillepich, A., Genel, S., Vogelsberger, M., Springel, V., Torrey, P., Rodriguez-Gomez, V., Sijacki, D., Snyder, G.F., Griffen, B., Marinacci, F., Blecha, L., Sales, L., Xu, D., Hernquist, L., 2015. The illustris simulation: Public data release. Astron. Comput. (ISSN: 2213-1337) 13, 12–37. doi:10.1016/j.ascom.2015.09.003, URL http://www.sciencedirect.com/science/article/pii/S2213133715000864.

Norman, M.L., Bryan, G.L., Harkness, R., Bordner, J., Reynolds, D., O'Shea, B., Wagner, R., 2007. Simulating cosmological evolution with enzo. doi:10.48550/arXiv.0705.1556, Astro-Ph.

Oliphant, T.E., 2015. Guide to NumPy, second ed. CreateSpace Independent Publishing Platform, USA, ISBN: 978-1-5173-0007-4.

Peebles, P.J.E., 1993. Principles of physical cosmology. doi:10.1515/9780691206721, Principles of Physical Cosmology By P.J.E. Peebles. Princeton University Press, 1993. ISBN: 978-0-691-01933-8, URL http://adsabs.harvard.edu/abs/1993ppc..book.....P.

Perez, F., Granger, B.E., 2007. IPython: A system for interactive scientific computing. Comput. Sci. Eng. (ISSN: 1521-9615) 9 (3), 21–29. doi:10.1109/MCSE.2007.53.

Potter, D., Stadel, J., Teyssier, R., 2017. PKDGRAV3: beyond trillion particle cosmological simulations for the next era of galaxy surveys. Comput. Astrophys. Cosmol. 4, 2. doi:10.1186/s40668-017-0021-1, URL http://adsabs.harvard.edu/abs/2017ComAC...4....2P.

Ragagnin, A., Dolag, K., Biffi, V., Cadolle Bel, M., Hammer, N.J., Krukau, A., Petkova, M., Steinborn, D., 2017. A web portal for hydrodynamical, cosmological simulations. Astron. Comput. 20, 52–67. doi:10.1016/j.ascom.2017.05.001, arXiv:1612.06380.

Revaz, Y., Jablonka, P., 2018. Pushing back the limits: detailed properties of dwarf galaxies in a LCDM universe. doi:10.1051/0004-6361/201832669, arXiv:1801.06222 [Astro-Ph].

Ross, R.B., Peterka, T., Shen, H.-W., Hong, Y., Ma, K.-L., Yu, H., Moreland, K., 2008. Visualization and parallel I/O at extreme scale. J. Phys. Conf. Ser. (ISSN: 1742-6596) 125, 012099. doi:10.1088/1742-6596/125/1/012099, URL https://iopscience.iop.org/article/10.1088/1742-6596/125/1/012099.

Schaller, M., Gonnet, P., Chalk, A.B.G., Draper, P.W., 2016. SWIFT: Using task-based parallelism, fully asynchronous communication, and graph partition-based domain decomposition for strong scaling on more than 100,000 cores. In: Proceedings of the Platform for Advanced Scientific Computing Conference. pp. 1–10. doi:10.1145/2929908.2929916, URL http://arxiv.org/abs/1606.02738 arXiv:1606.02738.

Schaller, M., Gonnet, P., Draper, P.W., Chalk, A.B.G., Bower, R.G., Willis, J., Hausammann, L., 2018. SWIFT: SPH with inter-dependent fine-grained tasking. Astrophys. Source Code Library ascl:1805.020 URL https://ui.adsabs.harvard.edu/abs/2018ascl.soft05020S.

Schaye, J., Crain, R.A., Bower, R.G., Furlong, M., Schaller, M., Theuns, T., Dalla Vecchia, C., Frenk, C.S., McCarthy, I.G., Helly, J.C., Jenkins, A., Rosas-Guevara, Y.M., White, S.D.M., Baes, M., Booth, C.M., Camps, P., Navarro, J.F., Qu, Y., Rahmati, A., Sawala, T., Thomas, P.A., Trayford, J., 2015. The EAGLE project: simulating the evolution and assembly of galaxies and their environments. Mon. Not. R. Astron. Soc. (ISSN: 0035-8711) 446, 521–554. doi:10.1093/mnras/stu2058, URL http://adsabs.harvard.edu/abs/2015MNRAS.446..521S.

Smith, A., Cole, S., Grove, C., Norberg, P., Zarrouk, P., 2022. A lightcone catalogue from the millennium-XXL simulation: improved spatial interpolation and colour distributions for the DESI BGS. doi:10.1093/mnras/stac2519, arXiv E-Prints arXiv:2207.04902.

Springel, V., 2005. The cosmological simulation code GADGET-2. Mon. Not. R. Astron. Soc. (ISSN: 0035-8711) 364, 1105–1134. doi:10.1111/j.1365-2966.2005.09655.x, URL http://adsabs.harvard.edu/abs/2005MNRAS.364.1105S.

The HDF Group, 1997. Hierarchical data format, version 5. https://www.hdfgroup.org/HDF5/.

Xiao, Q., Shang, P., Wang, J., 2012. Co-located compute and binary file storage in data-intensive computing. In: 2012 IEEE Seventh International Conference on Networking, Architecture, and Storage. pp. 199–206. doi:10.1109/NAS.2012.29.

Zheng, F., Zou, H., Eisenhauer, G., Schwan, K., Wolf, M., Dayal, J., Nguyen, T., Cao, J., Abbasi, H., Klasky, S., Podhorszki, N., Yu, H., 2013. FlexIO: I/O middleware for location-flexible scientific data analytics. In: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing. pp. 320–331. doi:10.1109/IPDPS.2013.46, ISSN: 1530-2075.