Monthly Notices of the ROYAL ASTRONOMICAL SOCIETY

MNRAS **531,** 1870–1883 (2024) https://doi.org/10.1093/mnras/stae894
Advance Access publication 2024 May 23

GUPPY I: a code for reducing the storage requirements of cosmological simulations

Philip Mansfield [1,2,3]★ and Tom Abel[1,2,3]

[1]*Kavli Institute for Particle Astrophysics and Cosmology, Stanford University, 452 Lomita Mall, Stanford, CA 94305, USA*
[2]*Department of Physics, Stanford University, 382 Via Pueblo Mall, Stanford, CA 94305, USA*
[3]*SLAC National Accelerator Laboratory, 2575 Sand Hill Road, Menlo Park, CA 94025, USA*

Accepted 2024 March 20. Received 2024 March 20; in original form 2023 July 19

ABSTRACT

As cosmological simulations have grown in size, the permanent storage requirements of their particle data have also grown. Even modest simulations present a major logistical challenge for the groups which run these boxes and researchers without access to high performance computing facilities often need to restrict their analysis to lower quality data. In this paper, we present GUPPY, a compression algorithm and code base tailored to reduce the sizes of dark matter-only cosmological simulations by approximately an order of magnitude. GUPPY is a 'lossy' algorithm, meaning that it injects a small amount of controlled and uncorrelated noise into particle properties. We perform extensive tests on the impact that this noise has on the internal structure of dark matter haloes, and identify conservative accuracy limits which ensure that compression has no practical impact on single-snapshot halo properties, profiles, and abundances. We also release functional prototype libraries in C, Python, and Go for reading and creating GUPPY data.

Key words: methods: numerical – software: public release – dark matter.

1 INTRODUCTION

In 2005, the hard drives required to store ten snapshots of the world's largest cosmological simulation would have cost $2800, after correcting for inflation.[1] Today, the equivalent task would cost $9900.[2]

As theoretical cosmology progresses, simulators have an ever-increasing demand for storage, even relative to the best available technology. Part of this demand comes from the grand scope of some modern theoretical projects. For example, the Abacus Summit simulation suite (Maksimova et al. 2021) aims to provide cosmology predictions for next-generation survey instruments such as the Dark Energy Spectroscopic Instrument (DESI) and the *Roman Space Telescope*, and thus spans a wide range of cosmologies while maintaining high mass resolution ($m_p \approx 2 \times 10^9\ h^{-1}\,\mathrm{M}_\odot$) across large scales ($L = 2\ h^{-1}$ Gpc). This leads to their public data release being over 2 PB in size, despite substantial subsampling. More generally, a combination of different Moore's law slopes and increased investment in computing mean that the speed of high-performance computing has been outpacing the decreasing cost of hard drive space for decades. Between 2005 and 2023, the cost-per-byte of storage has only decreased by a factor of thirty, but the combined speed of the 500 most powerful supercomputers has increased by a factor of more than two thousand.[3]

Cosmological simulations are expensive and time-consuming to run. Data restrictions substantially affect the community's ability to use the simulations we have invested so many resources in running and reduces the scientific output-per-dollar. Researchers will often need to heavily subsample their simulation data (e.g. Maksimova et al. 2021) or only store particles at a coarse snapshot cadences (e.g. Klypin, Prada & Comparat 2017). Subsampling makes it difficult to study the detailed small-scale structure of haloes and their subhaloes, while coarse output cadence prevents analysis that requires tracking particles over time, like merger tree construction (e.g. Srisawat et al. 2013), particle tracking (e.g. Diemer 2017), and some types of 'orphan' halo modelling (e.g. Guo et al. 2011). Some researchers will run detailed analysis 'on the fly', generating all their halo catalogues and data products as their simulations run and never outputting full snapshots (e.g. Angulo et al. 2012). Some *N*-body codes such as SWIFT (Schaller 2018) and GADGET-4 (Springel et al. 2021) build this feature in by default. While this process is the right choice for some projects, it locks researchers in to their initial analysis choices, makes it more difficult to explore unintended research directions, and completely prevents many forms of replication.

One of the greatest drawbacks of large simulation sizes is accessibility. There are relatively few well-funded groups which have access to the large data servers needed to comfortably work with simulation snapshots. In many research groups, a student who wanted

★ E-mail: phil1@stanford.edu

[1]Millenium Simulation (Springel et al. 2005). Median hard drive price from https://jcmit.net/diskprice.htm. Corrected for inflation according to the Consumer Price Index. Electricity and server maintenance not included.

[2]Uchuu (Ishiyama et al. 2021). Median hard drive price from https://jcmit.net/diskprice.htm. Electricity and server maintenance not included. Stanford RCC would currently charge about $1700 per month to store these data on its high-performance system.

[3]www.top500.org

to work with particle data would need to store that data on their personal computer, a task which is impractical for even modest data sets. This means that even though there is a wealth of publicly available simulation data (e.g. Klypin et al. 2017; Nelson et al. 2019; Maksimova et al. 2021; Villaescusa-Navarro et al. 2021), entire categories of analysis of this data are locked off from a substantial portion of our community due to data sizes.

One option for mitigating the impact of simulation sizes is data compression. Compression replaces the on-disc representation of a data set's integers, floats, and strings with an alternative representation that requires fewer bits (e.g. Sayood 2006). By leveraging the properties of their data set, researchers can develop algorithms which outperform generic compression tools in the same way that the JPEG format can store images more efficiently than `gzip` can after being applied to a raw image bitmap.

Computational biologists and chemists have used specialized compression algorithms to deal with the sizes of their *N*-body molecular dynamics simulations for decades. The general framework of this approach has been to modestly reduce accuracy, then to use a variety of tricks and data symmetries to store the lower accuracy values. This process is called 'lossy' compression, a technique that is widely used in image (e.g. Wallace 1992), audio (e.g. Sterne 2012), and video (e.g. Le Gall 1991) formats. Omeltchenko et al. (2000) was the first to develop a specialized lossy format for molecular dynamics data sets and later authors have developed a wide range of alternative compression algorithms (e.g. Meyer et al. 2006; Marais et al. 2012; Kumar et al. 2013; Huwald et al. 2016; Dvořák, Maňák & Váša 2020). In our view, none of these methods can be directly applied to cosmological simulations as-is. The most effective methods either re-order data, erasing information on particle identities (e.g. Omeltchenko et al. 2000), take advantage of the comparatively fine output cadences required by molecular dynamics analysis (e.g. Huwald et al. 2016), leverage physical symmetries in molecule-scale data that do not exist at larger scales (e.g. Dvořák et al. 2020), or are more appropriate for simulations with lower accuracy tolerances than most cosmological analysis requires (e.g. Meyer et al. 2006).

Compression algorithms have long been used in the compression of astrophysical images (e.g. Hanisch et al. 2001; Pence, White & Seaman 2010), and several authors have written compression algorithms directly targeting *N*-body data (Di & Cappello 2016; Tao et al. 2017; Li et al. 2018; Di et al. 2019; Cheng et al. 2020; Jin et al. 2020). While some cutting-edge algorithms perform very poorly on *N*-body data (see comparison in Tao et al. 2017), others claim very respectable compression ratios, ranging from a factor of four (Cheng et al. 2020) to a factor of twenty (Di et al. 2019). However, there is substantial uncertainty in what these compression ratios actually mean. The storage efficiencies of all lossy compression algorithms are strongly dependent on their accuracy, and there is no clarity in the literature on how much accuracy simulation outputs actually need. Many existing tests are run at accuracies which are too coarse for cosmological analysis (see Section 3) and the few tests on the impact of compression that do exist are not sensitive to the inner structure of haloes (see Section 3). There is also a wide disparity in the ways that different approaches control error sizes.

In addition to these more theoretical works, conservative compression techniques have been used in a few large-scale simulations. Abacus Summit (Maksimova et al. 2021) uses a pair of compression algorithms, RVINT and PACK9, which reduce positions and velocities to roughly 8.5 bytes per particle. TianNu (Emberson et al. 2017) uses a similar scheme which allows them to store both vectors with 9 bytes per particle. The Symphony zoom-in simulation (Nadler et al. 2023) suite stores only the floating-point mantissa of its positions and velocities, truncated at two bytes, giving positions, velocities, IDs, and other particle metadata at the cost of 12 bytes per particle. All three simulation teams err on the side of caution and store particles to much finer accuracies than are typically used by the dedicated compression literature. This is due to the issue mentioned above: there is no clear guidance on how much precision is needed.

This paper is essentially an argument to the cosmology community to begin applying these aggressive compression techniques to our data. First, we present the compression algorithm GUPPY in Section 2. This algorithm is based on old but effective compression techniques which have shown promise in recent tests (Di et al. 2019). More importantly, in Section 3, we perform an extensive suite of tests which demonstrate the impact of different compression accuracies on dark matter halo properties and recommend accuracy limits that ensure virtually no impact on halo properties. Lastly, in Section 4, we show that GUPPY can reduce the sizes of simulations by an order of magnitude at this accuracy level.

Upon the publication of this paper, we will release an open source implementation of the GUPPY algorithm in the programming language Go and provide libraries for using it in GO, C, and PYTHON.[4] This code is under active development and we aim to help support projects which wish to use it.

2 THE GUPPY ALGORITHM

The core of the GUPPY algorithm follows a well-worn pattern for compressing floating point data (see Sayood 2006 for a pedagogical discussion of this framework and Omeltchenko et al. 2000 for a classic application).

(i) The user chooses how accurately a variable should be stored. That variable is converted into a set of indices specifying a cell in a grid whose cell width is at least as small as the target accuracy. This is called 'quantization' (Section 2.2).

(ii) While compressing an array, GUPPY attempts to predict the value of the i^{th} element based on the elements 0 to $i-1$. Then the exact difference between this prediction and the true value is stored (Sections 2.1 and 2.3).

(iii) Lastly, a generic compression step, called 'entropy encoding' is performed on these offsets using an industry-standard compression library (Section 2.4).

This process is illustrated with a cartoon in Fig. 1. Overall, the structure of GUPPY is similar to the method discussed in Di et al. (2019), although in detail a number of parts of the two designs are different.

Throughout this paper, we assume that the compression target is a dark matter-only cosmological simulation, which natively stores positions and velocities as 3-vectors of 32-bit floats and stores particle IDs as 64-bit integers. We also assume that there is some unique mapping between a particle's location in the initial conditions grid (i.e. its Lagrangian coordinates). These assumptions are true for, e.g. a Gadget-2 simulation run with 2048^3 particles (Springel 2005a).

Simulations with fewer than 2^{32} particles can get away with using 32-bit IDs natively. To convert our reported compression ratios to these smaller simulations, multiply them by $1.143 = (3 \times 4 + 3 \times 4 + 8)/(3 \times 4 + 3 \times 4 + 4)$.

The GUPPY algorithm would also work on other particle properties, such as acceleration, density, or hydrodynamic variables. We do not

[4] https://github.com/phil-mansfield/guppy

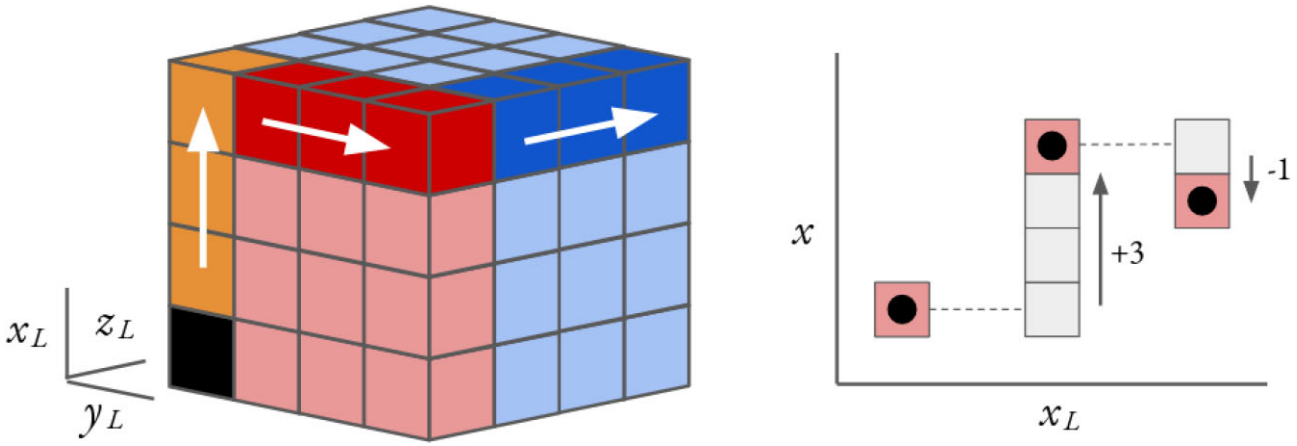

Figure 1. An illustration of the particle ordering used by GUPPY (left) and of delta encoding (right). Left: Data in GUPPY is arranged into 'skewers', lines of particles through Lagrangian space whose offsets relative to one another are encoded. Skewers are arranged to fill out the entire volume of a block of particles within Lagrangian space, while ensuring that all but the first contain a previously encoded/decoded particle immediately before the start of the skewer (Section 2.1). This ensures that only a single absolute offset needs to be stored. In this figure, the black cube in the lower left corner shows the particle whose value is stored directly. Next, the orange skewer (left-most arrow pointing upwards) is stored along the x_L direction in Lagrangian space, then the red skewers (middle arrow, pointing right) along y_L, then the blue (right-most arrow, pointing right) along z_L. The arrows indicate the direction that offsets are calculated along. Right: GUPPY 'quantizes' floating point values onto an integer grid (Section 2.2). GUPPY then predicts the value of the next integer given the previous one. and only records the difference between the prediction and actual value, the 'delta'. (Section 2.3). This figure illustrates a scheme where each particle is predicted to have the same value as the preceding particle in Lagrangian space. Our approach is similar in philosophy, but there have been several small modifications which improve compression ratios (see Section 2.4). Here, x_L stands in for whichever Lagrangian coordinate a skewer changes over, and x stands in for any particle property.

test this type of application in this paper, and recommend that users interested in doing so preform tests comparable to those in Section 3.

2.1 Data ordering

The core fact underpinning the GUPPY algorithm is that most particles stay relatively close to the particles that had been their neighbours in the initial conditions. For particles which have not fallen into collapsed structures, the mapping between initial ('Lagrangian') and final positions, $\vec{x}(\vec{x}_L)$ is a smooth and slowly changing function. Particles which fall into haloes or filaments will quickly separate from their old neighbours (see Hahn, Abel & Kaehler 2013; Sousbie & Colombi 2016, for a beautiful illustrations of this), but still stay localized within an object much smaller than the simulation box.

To take advantage of this, GUPPY stores particles in 'blocks', 1D arrays that contain all the particles within some 3D grid in the initial conditions. There are many elegant 1D orderings of grid cells (e.g. Butz 1971), however our testing showed that these orderings have worse compression properties than the substantially less elegant method we outline below when combined with the prediction method described in Section 2.3.

GUPPY's blocks are broken up into a number of 1D 'skewers' through the initial conditions grid. These skewers hold two Lagrangian coordinates fixed, while incrementing the third.

The skewer ordering that GUPPY uses is illustrated in the left panel of Fig. 1. Each small cube represents the location of a particle in the initial conditions, and the dark cubes show example skewers through this grid. First, the initial particle is stored (black). Next, a single skewer adjacent to the initial particle is stored (orange). Next, a face of the grid is filled out with skewers which are perpendicular to the direction of the initial skewer. Lastly, the body of the grid is filled out with skewers which are parallel to the normal of that face (blue). This ordering means that every skewer is stored later in the block than the particle immediately preceding it in the initial conditions grid.

Later encoding steps become more efficient if the mean change in a variable along a skewer is zero. Because of this, when GUPPY is storing x or v_x, the first (orange) skewer is in the $\hat{x}$ direction. Then a face is filled out in the $\hat{y}$ direction and the body of the block is filled out in the $\hat{z}$ direction. The direction of the first skewer is chosen analogously for the other positions and velocity variables.

Compression rates are not strongly dependent on the size of these blocks. Throughout this paper we use blocks containing 128^3 particles.

2.2 Quantization and dequantization

GUPPY handles lossy compression through a quantization step while writing data and a dequantization step when reading data. A data point, x_i is quantized to an index, d_i via

$$d_i = \lfloor x_i/\delta_x \rfloor, \tag{1}$$

where δ_x is the user-specified accuracy. Upon reading, d_i is dequantized back into x_i via

$$x_i = \delta_x(d_i + \mathcal{U}(0, 1)). \tag{2}$$

Here, $\mathcal{U}(0, 1)$ is a uniform random number between 0 and 1.

Quantization is the only step in GUPPY which loses information: all other steps are exact. This means that all errors induced by GUPPY are strictly bounded. Users can exactly simulate the impact of compression on their data easily without needing to run GUPPY by applying equations (1) and (2) to their data in sequence. Errors experienced by different particles are completely uncorrelated.

2.2.1 Alternative quantization schemes

There are many possible alternative quantization schemes. One could imagine, for example, storing vectors as a quantized norm and a pair of quantized angles (using, e.g. HEALPIX indices; Gorski et al. 2005), or using relative (i.e. split-logarithmic) errors instead of absolute

ones. Such alternative quantizations have been shown to lead to high compression ratios when applied to velocity vectors (Di et al. 2019; Cheng et al. 2020). In principle, GUPPY can accommodate these types of schemes if the user applies a remapping to input variables prior to compression, but we strongly urge against quantizing velocity vectors in this way.

None of the schemes mentioned above are invariant under changes in reference frame velocity. This means that the internal motions of particles inside haloes will be resolved with different accuracies depending on the halo's velocity relative to the arbitrary rest frame of the simulation. For example, small dark matter haloes with internal velocity dispersions $\lesssim 100\,\mathrm{km\,s^{-1}}$ can easily be found in large-scale bulk flows with velocities of 100s of $\mathrm{km\,s^{-1}}$ or orbiting cluster-mass haloes at 1000s of $\mathrm{km\,s^{-1}}$. In these cases, internal velocities will be more poorly resolved in the inertial frame of these small haloes and the resulting errors could be highly anisotropic. Additionally, a dependence on a halo's inertial frame would mean that compression errors would pick up correlations with physically meaningful quantities like large-scale environment.

In principle, the choice of pseudo-random number generator (PRNG) could impact the quality of the dequantization process. Numerous early PRNGs experienced highly non-random correlations in their output. The most famous of these failures is likely IMB's RANDU, a 60's-era linear congruential generator which shows obvious 'stripes' in its output when used to populate a 3D cube with points (e.g. Entacher 1998). The class of test which can detect these stripes within unit cubes of various dimensionalities – a 'spectral test' – is included as part of exhaustive modern PRNG test batteries (e.g. L'ecuyer & Simard 2007). GUPPY uses 'xorshift128+' (Vigna 2014) a variant of the xorshift class of PRNGs introduced in Marsaglia (2003). We chose this PRNG primarily due to its combination of speed and reliability, a combination which has made it see common use in numerous industry applications, particularly in the internal JavaScript engines for essentially all major web browsers.

TestU01 'BigCrush' (L'ecuyer & Simard 2007)[5] is an exhaustive set of 106 statistical tests of randomness for PRNGs and is generally considered to be the gold standard for PRNG reliability. There is some controversy over whether xorshift128+ actually passes all the BigCrush tests. Vigna (2014) claim that the algorithm passes the full test suite, while some reproducible, open source testing projects[6,7,8,9] find that it fails some tests (see also, Lemire & O'Neill 2019), particularly the `smarsa_MatrixRank` test (which tests the independence of columns in large random binary matrices) and the `scomp_LinearComp` test (which measures the complexity of a bit sequence based on the Lempel-Ziv compression algorithm). Whether or not xorshift128+ actually passes these higher order tests is not important for us: for our purposes it is only important that it generates uniform distributions within a 6D phase cube. This is well-established by other passed tests in the BigCrush suite, particularly the `snpair` series of tests, which checks properties of the nearest neighbour distribution for uniformly generated points in up to 12 dimensions, and the set of spectral tests performed by the suite. To our knowledge, these test results are not contested, except at scales far below the precision required by this paper (Haramoto & Matsumoto 2019).

[5] http://simul.iro.umontreal.ca/testu01/tu01.html
[6] https://github.com/lemire/testingRNG
[7] https://lemire.me/blog/2017/09/08/the-xorshift128-random-number-generator-fails-bigcrush/
[8] https://xoshiro.di.unimi.it/hwd.php
[9] https://www.pcg-random.org/statistical-tests.html

2.3 Delta encoding and decoding

To encode quantized d_i values, GUPPY attempts to predict the value of d_i using the previous $i-1$ values, and then stores the exact difference between its prediction and the true values, $\Delta p_i = d_i - p_i$.

The simplest method of this kind is referred to as delta encoding (e.g. Cherry 1953). In this case,

$$p_i = d_{i-1}. \tag{3}$$

In other words, the prediction for each value is identical to the previous decoded value. This is illustrated in the right figure of 1.

Surprisingly, our experiments showed that delta encoding modestly outperforms the compression ratios achieved by more complicated methods and substantially outperforms the speed of some such methods. We tested methods including linear extrapolation, $p_i = 2\,d_{i-1} - d_{i-2}$, extrapolating various polynomial fits to the previous several decoded points, storing the coefficients to a local fit to d_i, $f(x_L)$, for an entire skewer, then encoding $p_i = \lfloor f(x_{L,i}) \rceil$, as well as allowing GUPPY to switch between methods depending on how locally effective a prediction method was.

The unexpected success of delta encoding seems to be because predicting the location of particles in haloes is very difficult, and most of the bytes in a given GUPPY file are spent encoding halo particles rather than the easier-to-predict uncollapsed particles. This means that although all the aforementioned methods are wildly successful at compressing particles in this pre-collapse phase, that added efficacy has little impact on the total compression ratio. The failure mode of delta encoding is also less catastrophic than other methods, which counterintuitively leads to better performance in haloes. As such, GUPPY uses delta encoding during its prediction step.

2.3.1 Particle trajectories

In addition to the ID-based encoding which GUPPY uses, we also experimented extensively with trajectory-based approaches, where particle properties are predicted either by extrapolating particle properties from previous snapshots or by measuring deviations from some fit to particles' trajectories. This is the approach taken by the most effective molecular dynamics compression algorithms (Huwald et al. 2016) and can lead to impressive compression ratios in cosmological simulations, too (Li et al. 2018). While this is an extremely promising field of research, we did not use this class of approach for the initial release of GUPPY for the following reasons:

(i) Extrapolation-based compression requires that the reader open and analyse files in order, meaning that analysing a single snapshot near the end of a simulation can be a computationally expensive task. It also requires that users store all the preceding snapshots locally. This can result in an *increased* storage requirement for users who are only interested in a single snapshot and therefore does not necessarily help data accessibility. This issue can be somewhat mitigated by intermittently storing non-extrapolated snapshots which can be used as the starting point for several following extrapolated snapshots (Li et al. 2018), but even this approach requires that users always store at least one non-extrapolated snapshot, which limits the savings these methods can achieve on low-resource systems.

(ii) Fit-based approaches have fast random-access times, but need to amortize the cost of storing fit parameters across many snapshots

to realize their space savings. So this method also suffers from accessibility issues for low-disc-space users.

(iii) Trajectory-based approaches are fantastically effective at compressing particles in the linear pre-collapse regime, but effectively compressing particles once they experience strong accelerations within halo centres requires fairly fine snapshot cadence. For a particle on a circular orbit, velocity-based extrapolation only outperforms randomly picking a point within the particle's orbit when snapshot cadences are smaller than $\approx 1.75\, T_{\rm orbit}/2\pi$. For a simulation with the fine output cadence of IllustrisTNG, this will be true for most orbits smaller than $\approx R_{\rm vir}/5$ of their respective haloes and for a simulation with an output cadence similar to AbacusSummit, this will occur at approximately $\approx R_{\rm vir}/2$.

2.4 Entropy encoding

After a block of simulation data has been converted to an array of small integers, GUPPY applies an 'entropy encoding' step which represents those integers in the smallest amount of space possible. The bulk of this work is done by an industry-leading compression library called ZSTANDARD.[10] ZSTANDARD was developed primarily by Yann Collet and was designed to be a faster alternative to ZLIB, the library used to create `.gzip` files.

ZSTANDARD reads in a stream of bytes and breaks it up into a set of 'literals:' Literals are individual bytes or sequences of bytes that the library will represent directly. ZSTANDARD constructs a data structure called a 'Huffman tree' (Huffman 1952) which assigns each literal a code based on its frequency. More common literals are given shorter codes. The much older Morse Code relies on a similar principle: common letters like E and T are only one keystroke, but uncommon letters like X and Z are four keystrokes. Once the set of literals is fixed, a Huffman tree produces the most space-efficient coding possible given their frequency in a data set. ZSTANDARD uses an extremely efficient Huffman tree library, HUFF0.[11]

Once ZSTANDARD has converted its literals into the optimal set of codes, it identifies literals that repeat several times, called 'sequences' and represents them as a single literal and the number of repetitions. This is called 'run-length encoding' (Cherry et al. 1963).[12] Lastly, ZSTANDARD further encodes all its own internal data structures – such as its sequences and Huffman trees – with Finite State Encoding, an algorithm whose theory was developed by Duda (2013) and which is implemented by the FSE library.[13]

Qualitatively, ZSTANDARD is most effective at compressing data where *some values appear much more frequently than others* and where *the same values are repeated many times in a row*. The same is true for almost all major compression libraries.

The simplest way to compress the array of Δp_i values produced by the delta encoding discussed in Section 2.3 would be to convert the array of integers to bytes and feed those bytes to ZSTANDARD. However, this approach produces very poor compression. Even though $|\Delta p_i|$ is small, half of all Δp_i values are negative. Negative integers are represented using two's complement, meaning that most of their higher order bits are 1's, instead of 0's. This means that the byte stream received by ZSTANDARD is mostly small sequences of `0x00` and `0xff` bytes which flip back and forth between one another almost at random, interspersed by small amounts of 'real' data. In practice ZSTANDARD (and all other entropy encoding libraries we tested) cannot effectively take advantage of the structure in this data.

To rectify this, we add two pre-processing steps. First, we add an offset, P_0 to Δp_i. This is common practice when delta encoding, and usually $P_0 = \min(\Delta p_i)$ to ensure that all the deltas are positive. However, if $\mathrm{mode}(\Delta p_i) \bmod 128$ is close to 0 or 127, roughly half of all Δp_i values will have one value for the second-least-significant byte and the other half will have a different value. To avoid this effect, we scan the histogram of Δp_i values and find a value $\Delta p_{\rm window}$ which maximizes the sum

$$\sum_{j=\Delta p_{\rm window}}^{\Delta p_{\rm window}+127} |\{\Delta p_i = j\}|. \tag{4}$$

We then set

$$P_0 = \min(\Delta p_i) + ((\min(\Delta p_i) - \Delta p_{\rm window}) \bmod 128). \tag{5}$$

This ensures that the second-least-significant bytes of the integers given to ZSTANDARD stay fixed for the overwhelming majority of Δp_i values. Adding this step improves compression ratios at the few-per cent level. This step also ensures that if there is an average, non-zero offset between particles and their predicted values along the encoding order, it will have no effect on final compression ratio, so we do not need to permute the dimensional ordering of skewers when compressing different components of vector quantities.

Lastly, we transpose the bytes of the integer array so that all least-significant bytes are consecutive, all second-least-significant bytes are consecutive, etc. Each set of bytes is compressed separately. This allows ZSTANDARD's run-length encoding to compress the unchanging higher order bytes by factors of $\approx$ thousands and prevents the values that these bytes hold from polluting the literal frequencies used to construct the Huffman tree which encodes the lower order bytes.

Entropy encoding is GUPPY's most computationally expensive step, with almost all of that time being spent in ZSTANDARD. A comparison of ZSTANDARD against other leading entropy encoders shows that at a fixed compression ratio, ZSTANDARD outperforms most of its competitors,[14] so this performance bottleneck would exist regardless of which particular library we used.

2.5 Public release of code

Upon publication, the publicly available source code for GUPPY 1.0.0 will be found at https://github.com/phil-mansfield/guppy. This release will include a command line program which converts Gadget snapshot files to Guppy files, libraries for reading Guppy files in C, PYTHON, and GO. We have also added support for GUPPY files to the ROCKSTAR halo finder (Behroozi, Wechsler & Wu 2013).[15]

Many applications require detecting whether the particles in a file overlap with a particular region of space. Unlike many other simulation output files, the Lagrangian ordering of GUPPY files sometimes causes the files to span across the periodic boundaries of simulation boxes or have widths that are larger than half the simulation box size. Both facts can cause commonly used algorithms for detecting the overlap between two regions to fail, as these

[10] https://github.com/facebook/zstd
[11] https://github.com/Cyan4973/FiniteStateEntropy
[12] Run-length encoding has probably been in use as a bookkeeping tool since the invention of writing itself. The cited example of it is the first time that it was applied to compress electronic signals that we could identify.
[13] https://github.com/Cyan4973/FiniteStateEntropy
[14] https://quixdb.github.io/squash-benchmark/
[15] This is currently done through the fork https://bitbucket.org/philip-mansfield/rockstar/src/main/

algorithms often implicitly assume that both regions are smaller than half the box size. We briefly describe an algorithm for correctly computing bounding boxes in Appendix A. We recommend that users adapting their code to work with GUPPY use these algorithms instead of their existing bounding box functions.

3 COMPRESSION ACCURACY

In this section, we perform a battery of tests designed to determine accuracy limits on positions and velocities, δ_x and δ_v which are small enough to prevent biasing most research performed on simulation snapshots. As a review of Section 2.3, particles are effectively discretized to a 6D grid with cell widths of δx in the x dimensions and δv in the v dimensions. When a particle is decompressed, it is given a random value within that cell. This means that δ_x and δ_v are maximum bounds on the error in each dimension, not average values. The maximum total errors in $|\vec{x}|$ and $|\vec{v}|$ are $\sqrt{3}\delta_x$ and $\sqrt{3}\delta_v$, respectively.

We choose to focus on the small-scale properties of dark matter haloes and on subhalo statistics, as these are both commonly used and have strict accuracy requirements. Researchers interested in the larger scale distribution of matter could comfortably use coarser accuracies.

3.1 Simulations and external codes

To study the accuracy and compression efficiency of GUPPY, we use a number of simulations and public codes.

In this paper, we primarily use the dark matter-only simulation Erebos_CBol_L125 (Diemer & Kravtsov 2015). Erebos_CBol_L125 is a 1024^3-particle Gadget-2 simulation (Springel 2005b) with $L = 125\ h^{-1}\,\mathrm{Mpc}$, $m_p = 1.36 \times 10^8\,h^{-1}\,\mathrm{M}_\odot$, a comoving Plummer-equivalent force softening scale $\epsilon = 2.4\,h^{-1}\,\mathrm{kpc}$, and timestepping parameter `ErrTolIntAcc` $= 0.025$. It was simulated with the same WMAP-like cosmology as the Bolshoi simulation (Klypin, Trujillo-Gomez & Primack 2011), $\Omega_m = 0.27$, $\Omega_\Lambda = 0.73$ $\Omega_b = 0.047$, $\sigma_8 = 0.82$, and $h = H_0/(100\ \mathrm{km\ s^{-1}}) = 0.70$. It contains a hundred snapshots logarithmically spaced between $z = 20$ and $z = 0$. In a few places, we also use other boxes in the Erebos suite, Erebos_CBol_L63, Erebos_CBol_L250, Erebos_CBol_L500, and Erebos_CBol_L1000 (Diemer & Kravtsov 2015). These boxes are configured similarly to Erebos_CBol_L125, but with $m_p = 1.70 \times 10^6\,h^{-1}\,\mathrm{M}_\odot$, $1.09 \times 10^9\,h^{-1}\,\mathrm{M}_\odot$, $8.79 \times 10^9\,h^{-1}\,\mathrm{M}_\odot$, and $7 \times 10^{10}h^{-1}\,\mathrm{M}_\odot$, respectively, and $\epsilon = 1\,h^{-1}\mathrm{kpc}$, $5.8\,h^{-1}\mathrm{kpc}$, $14\,h^{-1}\mathrm{kpc}$, and $33\,h^{-1}\mathrm{kpc}$, respectively. Each Erebos box totals 3.3 TB of data per simulation.

We also perform some analysis on ROCKSTAR halo catalogues. ROCKSTAR is a halo finder which identifies halo centres by finding 6D friends-of-friends groups in phase space with successively smaller and smaller linking lengths (Behroozi et al. 2013). We analyse maximum circular velocities, $V_{\rm max}$, NFW concentrations, $c_{\rm vir}$, Bullock spin parameters, λ, normalized offsets between a halo's centre of mass and its most bound particle, $x_{\rm off}$, short-to-long ellipsoidal axis ratios, c/a, and virial ratios, $|T/U|$. Readers can find a review of how the latest version of ROCKSTAR calculates these properties in section 2.4 of Mansfield & Avestruz (2021).

Throughout this paper, we use the Bryan & Norman (1998) definition of the virial radius, $R_{\rm vir}$, and the corresponding definitions of the virial mass, $M_{\rm vir} = (4\pi/3)\rho_{\rm vir}R_{\rm vir}^3$ and virial circular velocity, $V_{\rm vir} = \sqrt{GM_{\rm vir}/R_{\rm vir}}$. Note that at a fixed redshift, all haloes of the same virial mass have the same virial velocity.

In some places, we use the quantity

$$V_{\rm vir,100} = \sqrt{GM_{\rm vir}(N_{\rm vir} = 100)/R_{\rm vir}(N_{\rm vir} = 100)}, \quad (6)$$

i.e. the virial velocity of a hundred-particle halo. This is used to provide a characteristic velocity scale for simulations which scales with resolution.

3.2 Calculating halo properties

In Section 3.3 we will study the impact of δ_x and δ_v on enclosed mass profiles, $V_{\rm circ}(< r)$ shape profiles, $(c/a)(r)$, angular momentum profiles, $\vec{L}(r)$, and bound mass profiles, $f_{\rm bound}(r)$. In this section, we review the details of how these profiles are calculated.

We only calculate the first three properties on bound particles. To estimate boundedness, we compute the kinetic energy of particles, KE, in the frame of the halo, as estimated by ROCKSTAR. We then use a KD-tree[16] to compute the potential energy, PE, of particles. Particles are removed from analysis if KE $>|$PE$|$. We note that this is not the correct procedure to estimate the boundedness of particles in subhaloes as it does not account for external tidal fields, but we use it anyway to emulate the behaviour of halo finders which almost universally do not include the impact of tides in their boundedness calculations.

Enclosed mass profiles are parametrized as $V_{\rm circ}(< r) = \sqrt{GM(< r)/r}$. Shape profiles are computed by computing the shape tensor from particles within a single spherical shell and using the eigenvalues of that tensor to find the short-to-long axis ratio, c/a, of an ellipsoidal shell with the same shape tensor. Angular momentum is calculated by adding the angular momentum vectors of all the particles in a spherical shell. Bound fractions are calculated by finding $N_{\rm bound}/(N_{\rm bound} + N_{\rm unbound})$ within a given spherical shell.

For simplicity, we do not iteratively remove particles after unbinding and do not iteratively recompute shape tensors within our spherical shells. However, if single-iteration boundedness fractions and shapes before and after compression are nearly identical, the fully iterative boundedness fractions and shapes will also be very similar.

3.3 Profile accuracy

Before performing statistically rigorous analysis on the impact of compression on large populations of haloes, we first perform qualitative analysis on the impact of compression on the detailed properties of individual haloes which. As we will show, limits on compression accuracy that one would infer from these individual haloes similar to the limits one would infer from analysis of large halo populations. It is important to consider individual haloes first because if one were only to study the averages of large populations of haloes, one runs the risk that compression increases noise on an individual halo-by-halo basis, but does not change population averages.

In Fig. 2 we show the impact of different δ_x values on the circular velocity and enclosed shape profiles of a $\approx 10^6$ particle dark matter halo. In the left panel, the grey contour shows an estimate of the systematic bias already baked into the rotation curve by the simulation's force softening according to the simulation-calibrated models in Mansfield & Avestruz (2021) (using simulation data from Ludlow, Schaye & Bower 2019). No equivalent model exists for $c/a(< r)$, so instead we shade all radii smaller than 4ϵ,

[16] https://github.com/phil-mansfield/gravitree

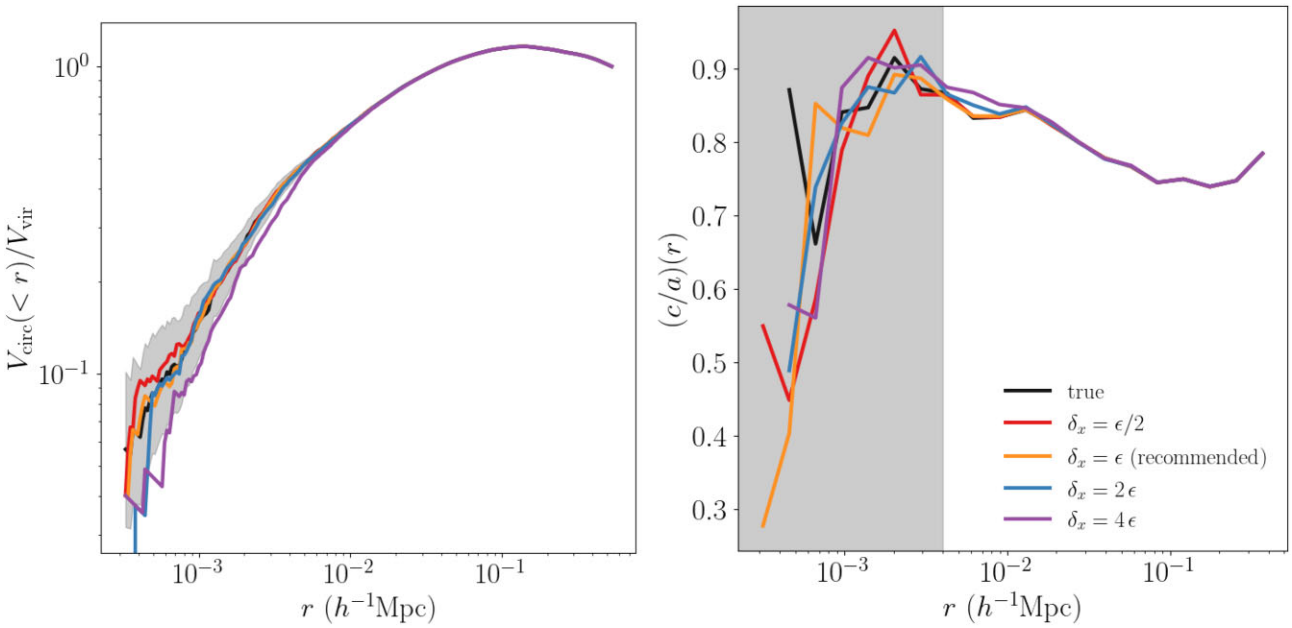

Figure 2. The impact of position accuracy on $V_{\rm circ}(r)$ (left) and shape, $(c/a)(r)$, (right) profiles. A single halo with $\approx 10^6$ particles is shown from a simulation with $\epsilon = 1$ km s^{-1}, Erebos_CBol_L63. The black lines show the true profiles of the halo and the coloured lines show increasingly coarse spatial accuracies in ascending rainbow order. The grey shaded region in the left panel shows the level of systematic bias injected into the circular velocity profile by force softening (Mansfield & Avestruz 2021), and the grey shaded region in the right-hand panel shows $4\,\epsilon$. Setting $\delta_x \lesssim 2\epsilon$ has barely any impact on the halo and what little impact it does have is restricted to regions which are already unreliable for scientific analysis. Our recommended accuracy of $\delta_x = \epsilon$ is comfortably below this limit. See Section 3.3 for discussion.

the approximate radius where biases in $V_{\rm circ}(< r)$ start becoming significant. This is a conservative choice for our analysis (which becomes more restrictive with smaller convergence regions), as Mansfield & Avestruz (2021) showed that force softening scales has a larger impact on $(c/a)(< R_{\rm vir})$ than it does on $V_{\rm max}$.

For this halo, compression with $\delta x \lesssim 2\epsilon$ has less impact on the mass profile than the intrinsic bias due to force softening. This is because force softening reduces central densities far beyond the radius where gravitational forces are altered (see discussion in Mansfield & Avestruz 2021), while the scales that compression impacts is more strictly bounded. We repeated this measurement for forty randomly chosen haloes with particle counts ranging from 100 to 10^6, and the level of bias shown here is typical. In a few cases, $\delta_x = 2\epsilon$ led to noticeable biases in $V_{\rm max}$, but $\delta_x = \epsilon$ was always well below existing biases. We defer a statistically rigorous analysis of this to the next section, but chose the halo in Fig. 2 because it was qualitatively representative of the full sample.

The shape profile shows a similar story. The halo becomes rounder at $r \lesssim 2\delta_x$ because compression flattens small-scale anisotropies. As before, the results for this halo are typical among forty randomly chosen test haloes, with a few outliers which required $\delta_x \lesssim \epsilon$.

The sudden downturn in the full-accuracy $(c/a)(< r)$ profile at small radii is likely caused by numerical effects. The small number of particles at these radii allows the shape tensor to find low-axis ratio configurations purely by chance. We see this behaviour in all our test haloes, although the radius where it occurs can change depending on the inner particle number density. The tests in Mansfield & Avestruz (2021) suggest that unrelated, force-softening-based biases likely extend to much larger radii than this down-turn.

Inaccuracies in velocities have little impact on mass and shape profiles, so to study the impact of δ_v, we focus on the angular momentum profiles and bound mass profiles of subhaloes rather than massive central haloes. Velocity information is used directly in computing both profiles and is critical for differentiating host particles from subhalo particles. We show $|\vec{L}(r)|$ and $f_{\rm bound}(r)$ for a representative subhalo in Fig. 3. We choose to illustrate the effect with this subhalo because its $f_{\rm bound}$ profile is strongly dependent on radius, implying that velocity information is more important for correctly resolving its structure than it would be for a more fully bound subhalo.

Compression has little impact on either profile, even as δ_v approaches $V_{\rm vir}/2$. This is because both quantities are computed from averages across many particles and quantization does not bias velocities low or high. However, when very large δ_v thresholds are used, additional noise appears in the $\vec{L}(r)$ profile. Therefore, these profiles do not place strong constraints on δ_v other than that $\delta_v \lesssim V_{\rm vir}/4$ can prevent excess noise in some quantities. These profiles are typical among the 40 random subhaloes we tested.

3.4 Halo catalogue accuracy

In addition to a qualitative study of individual objects, we look at the statistical impact of compression on full halo catalogues. We added support for GUPPY files to the ROCKSTAR halo finder (Behroozi et al. 2013) and generated halo catalogues for a range of δ_x and δ_v. Throughout this section, we focus on the simulation Erebos_CBol_L125. We have chosen this simulation because its convergence properties can be measured directly against the higher resolution Erebos_CBol_L63, and because its halo properties have been empirically shown to converge at relatively low particle counts compared to similar cosmological boxes (Mansfield & Avestruz 2021). This makes Erebos_CBol_L125 a particularly strict test box because compression-induced biases in halo properties are only meaningful when those properties are otherwise converged.

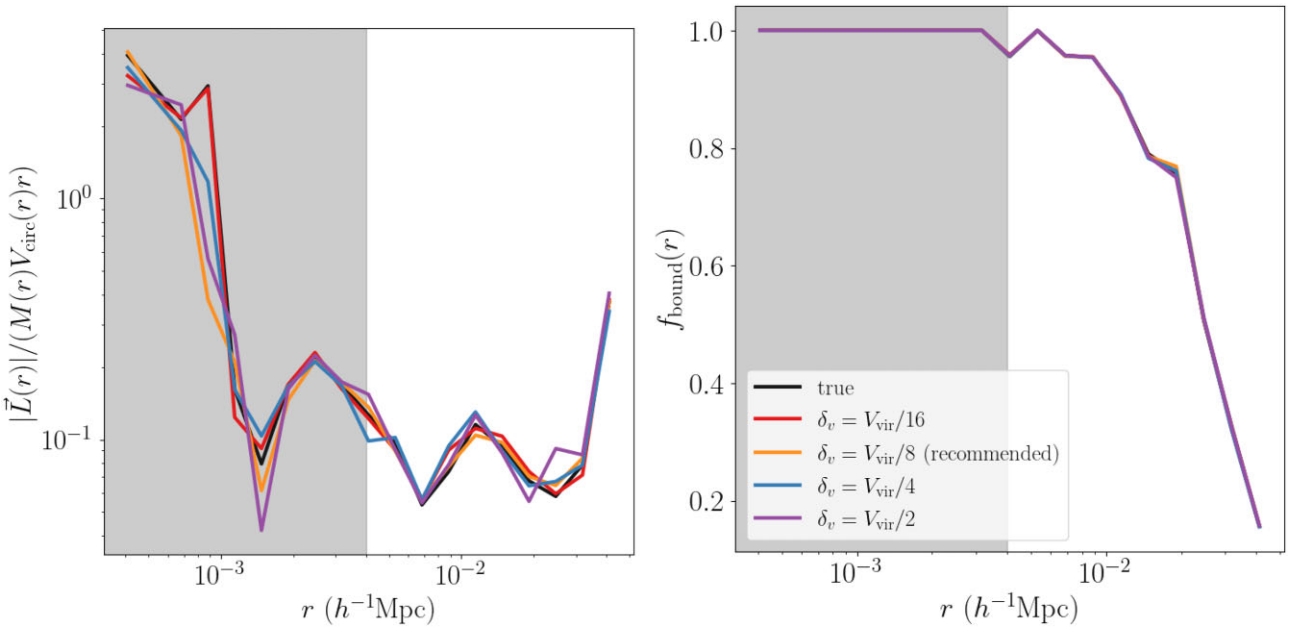

Figure 3. The same as Fig. 2, except scaled angular momentum (left) and bound mass fraction (right) profiles are shown and the curves correspond to different velocity accuracies, scaled by the virial velocity of the target halo. The halo shown here is a 1000-particle subhalo, chosen so $f_{\rm bound}$ varies strongly with radius. Even δ_v values far larger than the values we recommend for low-resolution haloes have little impact on halo properties. Note that our recommendation is actually more conservative than shown here: we recommend that 100-particle haloes are compressed to $\delta_v = V_{\rm vir}/8$, not 1000-particle haloes. However, such poorly resolved haloes have profiles which are too noisy to visually evaluate in the matter shown in this figure.

Fig. 4 shows the subhalo mass functions of group-mass haloes ($10^{13}\,h^{-1}\,{\rm M}_\odot < M_{\rm vir} < 10^{14}\,h^{-1}\,{\rm M}_\odot$). This subhalo mass function is shown down to 50-particle haloes, the smallest objects available in our catalogues. Historically, authors have argued for subhalo mass function convergence limits as low as 100 particles (e.g. Springel et al. 2008), but modern idealized simulations have demonstrated that the true convergence limit for subhalo abundances which are vastly larger than this (e.g. Errani & Navarro 2021, and references therein). We compare the subhalo mass function measured from the original uncompressed simulation against simulations which have various compression-induced spatial resolutions (left) and velocity resolutions (right).

We use the same host centres and radii across the different catalogues, regardless of what the masses and positions of the hosts are in the compressed catalogue. We do this because we want to study biases which are much smaller than the stochastic noise associated with a single realization of the subhalo mass function within a box of this size. Compression can slightly alter the masses of host haloes stochastically, which can result in a small number of objects leaving or entering the studied mass range. These sorts of changes in host sample can lead to small correlated – but entirely random – errors across different subhalo mass bins. This could give the impression that the compression itself is applying a small bias to subhalo abundance.

Compression has little impact on subhalo abundances. $\approx$ 1 per cent biases can only be seen at low masses for very coarse accuracies, $\delta_x \gtrsim 4\epsilon$ and $\delta_v \gtrsim V_{\rm vir,100}/2$. At $\delta_x = \epsilon$ and $\delta_v = V_{\rm vir,100}/8$, biases are kept well below the per-cent level. Compression injects some unbiased noise into the subhalo mass function relative to the original, but this noise is much smaller than the shot noise already present in the subhalo mass function. As such, this noise will have little impact on the number of hosts needed to bring fluctuations below some target analysis level.

We also manually inspected the positions and radii of the subhaloes within ten 10^6-particle haloes to test whether compression had a substantial impact on individual subhalo populations. We found that for the most part, subhaloes found after compression maintain very similar masses and positions to what they had before compression. This is expected, given that compression noise is far below the Poissonian level.

We find that compression *does* impact subhaloes which are likely to be numerical artefacts. ROCKSTAR will occasionally generate a fake – but large – subhalo which overlaps with the centre of its host and will occasionally identify subhaloes in the tidal tails of larger disrupting objects. See Srisawat et al. (2013) (section 5.3) Mansfield et al. (2023) (section 6.2 and appendix B) for extended discussions on tree errors and spurious haloes. These objects fluctuate in-and-out of existence from snapshot to snapshot and compression can sometimes change whether an artefact does or does not appear in particular snapshot for a particular halo. We found no evidence that compression increases or decreases the frequency of these artefacts, but note that these false haloes seem particularly sensitive to the exact positions and velocities of their particles and thus artefacts that appear in catalogues generated from full particle snapshots may be missing in GUPPY-generated catalogues and visa versa. We recommend that researchers remove these objects from their halo catalogues anyway,[17] so their exact frequency is not important. Compression can also affect haloes very close to the particle limit: a small increase or decrease in $N(< R_{\rm vir})$ can cause a halo to fluctuate above or below the limit of the catalogue. We found that both effects occur even when particles are randomly perturbed by amounts many

[17]For those unfamiliar, this removal can be done by tracing the main branches of all subhaloes back to their first snapshots and checking whether the object was already a subhalo at this point.

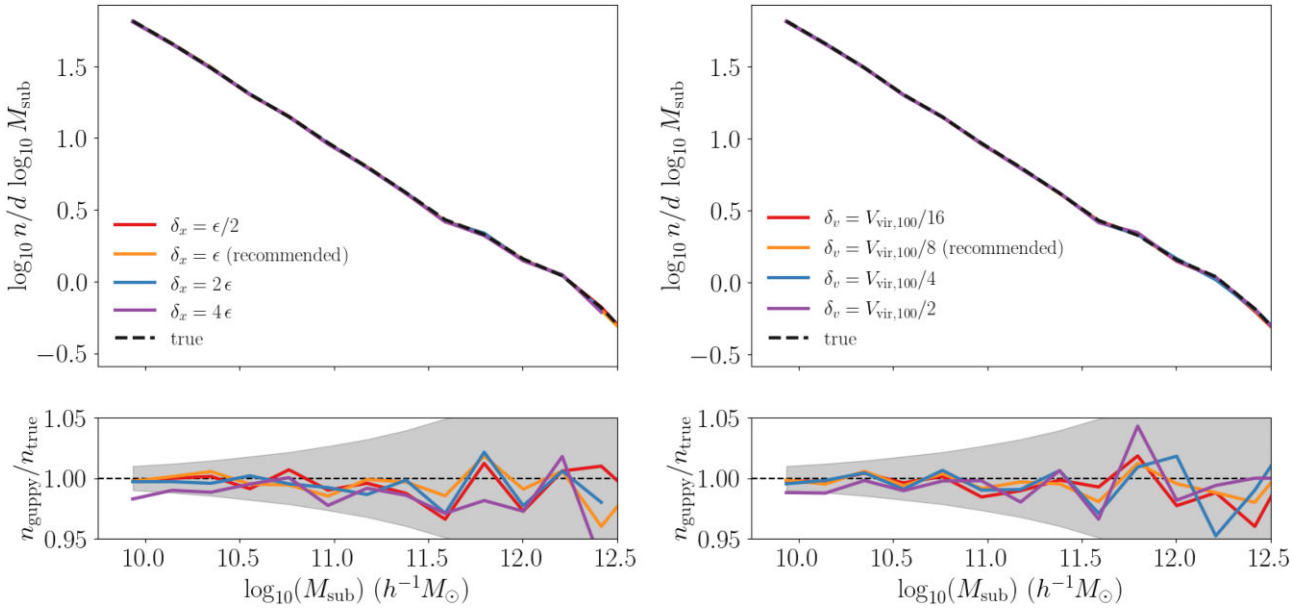

Figure 4. The impact of compression on subhalo abundances around group-sized haloes in Erebos_CBol_L125. Both panels show the differential subhalo mass function around host haloes with masses between $10^{13}\,h^{-1}\,\mathrm{M}_\odot$ and $10^{14}\,h^{-1}\,\mathrm{M}_\odot$. The left panels show the subhalo mass function from the original simulation as a dashed black line and the mass subhalo mass function measured in compressed catalogues with different spatial resolutions as coloured curves. The top panel shows the average number of subhaloes per host halo and the bottom panel shows the fractional difference in this average between each compressed simulation and the original. The grey shaded contour shows the 1-σ uncertainty on subhalo counts from uncorrelated shot noise. At $\delta_x \lesssim 2\epsilon$, there is little systematic bias in subhalo abundances and the added noise is highly subdominant to Poissonian noise. The right panel shows the same quantities but at different velocity compression levels. Similarly, the impact of compression on subhalo abundances is negligible when $\delta_v \lesssim V_{\rm vir,100}/4$. In the left panel, velocities are compressed to $\delta_v = V_{\rm vir,100}/32$ and in the right panel positions are compressed to $\delta_x = \epsilon$.

times smaller than the δ_x and δ_v ranges shown in Fig. 4, so we suspect that this stochasticity is mostly coming from fragility in the halo finder to subresolution noise.

Internal halo properties are more sensitive to compression resolution. Fig. 5 shows the biases in halo properties which are induced by different compression-induced spatial resolutions. For this figure, δ_v is set to a small value of $\delta_v = V_{\rm vir,100}/32$ to isolate the impact of position resolution. Fig. 5 shows the convergence limits for Erebos_CBol_L125 from (Mansfield & Avestruz 2021), which effectively means that they were estimated from comparison against Erebos_CBol_L63. Grey contours show 1-σ errors on the different halo properties, as estimated by jackknife-resampling performed on eight equal-volume sub-cubes.

Compression at $\delta_x \leq \epsilon$ has virtually no impact on the average values of halo properties at converged mass ranges. Even compression at $\delta_x = 2\,\epsilon$ generally leads to biases which are at the $\approx$ per-cent-level even at low masses. This is likely because most haloes are unaffected by compression at $\delta_x = 2\epsilon$, with only a few outliers seeing noticeable effects (Section 3.3). Bin-to-bin scatter for $\delta_x \leq \epsilon$ is substantially smaller than sample variance, meaning that bias-free noise added by compression is unlikely to be important for most analysis.

Based on Fig. 5 and our earlier analysis in Fig. 2, we recommend

$$\delta_{x,\rm rec} = \epsilon \tag{7}$$

as a conservative spatial resolution.

Fig. 6 shows similar analysis which investigates the impact of velocity resolution on halo properties. In this case, the different colours show different values of δ_v, with spatial resolution fixed to our recommendation of $\delta_x = \epsilon$. Only properties with an explicit dependence on a halo's kinetic energy experience meaningful δ_v-induced biases. For converged haloes, these biases are at the $\approx$ per-cent-level for $\delta_v \leq V_{\rm vir,100}/4$ and are essentially negligible for $\delta_v \leq V_{\rm vir,100}/8$. The bin-to-bin scatter induced by compression is negligible compared to the existing noise in the measurement due to sample variance.

Based on Fig. 6 and our earlier analysis in Fig. 3, we recommend

$$\delta_{v,\rm rec} = V_{\rm vir,100}/8 \tag{8}$$

as a conservative velocity resolution.

3.5 Merger trees

The previous sections only consider the accuracy of compression in a single snapshot, but many forms of analysis focus on the evolution of haloes over time. We consider a full test of the intersection between time evolution and compression to be outside the scope of this work due to its complexity, but note that there are several a priori reasons to expect that compression limits sufficient for single-snapshot analysis will also be sufficient for time-dependent (i.e. merger-tree-based) analysis. There are two types of halo evolution we should consider: the evolution of a central halo and the evolution of a subhalo.

For central haloes, creating a merger tree is usually relatively easy. An isolated halo that is not undergoing a significant merger will still contain most of its particles in the next snapshot, meaning that even very simple methods based on particle count cross-correlation or most-bound particles will be able to select the correct descendant halo (Srisawat et al. 2013). Quantization at our recommended levels is unlikely to alter this, as δ_x is too small to perturb a deeply bound particle out of its host halo or lead to a meaningful amount of particle flow into/out of a halo's virial radius.

Major mergers may be a different story. During nearly equal-mass mergers, the fundamental approximation of all halo finders –

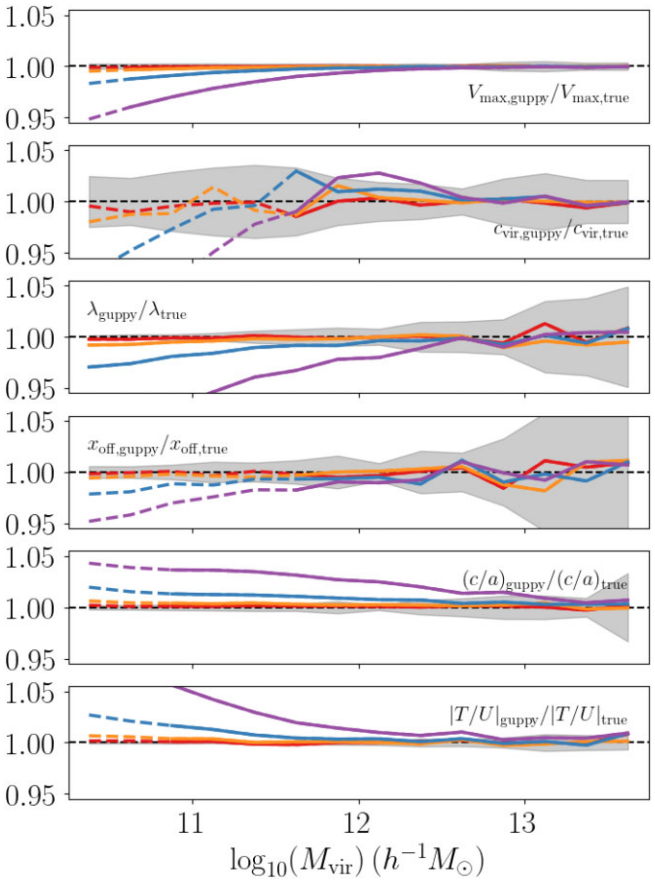

Figure 5. The impact of compression on halo properties. Each panel focuses on a different halo property, X, and shows the ratio $\langle X\rangle_{\rm guppy}/\langle X\rangle_{\rm true}$: the ratio of the mean of a property measured from compressed catalogues, $\langle X\rangle_{\rm guppy}$, to the original full-accuracy simulation, $\langle X\rangle_{\rm true}$. The ratios are shown as different colours for different spatial resolutions using the same colour scheme as the left panel of Fig. 4: red is $\delta_x = \epsilon/2$, orange is $\delta_x = \epsilon$, blue is $\delta_x = 2\,\epsilon$, and purple is $\delta_x = 4\,\epsilon$. 1-σ jackknife-estimated errors on the averages are shown as grey contours. The ratios are shown as dashed lines below the convergence limit for X and solid lines above it, as given by Mansfield & Avestruz (2021). Compression at $\delta_x \leq \epsilon$ has almost no impact on these averages at converged mass ranges. Even compression at $\delta_x = 2\epsilon$ only introduces ≈per-cent-level biases in most properties at low masses. The orange curve shows our recommended spatial resolution, $\delta_x = \epsilon$.

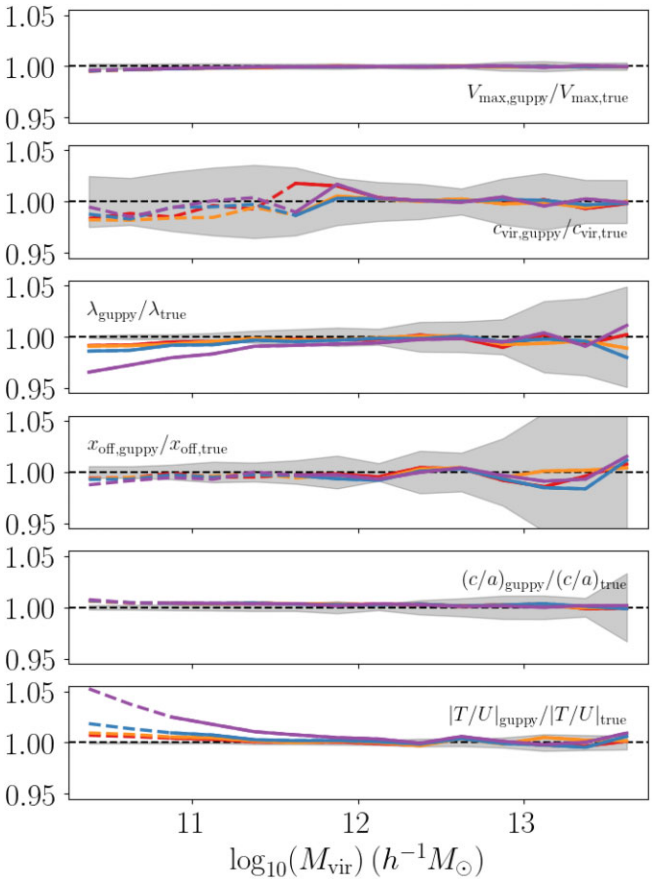

Figure 6. The same as Fig. 5, except the coloured curves show varying velocity resolutions instead of spatial resolutions. Colours have the same meaning as in the right panel of Fig. 5: red is $\delta_v = V_{\rm vir,100}/16$, orange is $\delta_v = V_{\rm vir,100}/8$, blue is $\delta_v = V_{\rm vir,100}/4$, and purple is $\delta_v = V_{\rm vir,100}/2$. Velocity compression has little effect on most properties and is restricted to quantities with explicit dependencies on the internal kinetic energy of the halo, such as λ and $|T/U|$. Even very coarse velocity resolutions have a minimal impact on halo properties, with biases for $\delta_v < V_{\rm vir,100}/4$ largely staying at the sub-per-cent level. The orange curve shows our recommended velocity resolution, $\delta_v = V_{\rm vir,100}/8$.

that haloes are roughly spherical objects which can be analysed in isolation – breaks down and this causes many modern halo finders similarly break down in this regime (Behroozi et al. 2015). This often leads to unpredictable switching of mass between the central halo and its secondary as relatively small snapshot-to-snapshot changes propagate out into large inferred changes in the properties of the joint system (Mansfield & Kravtsov 2020; Wang et al. 2023). We find it plausible that this sensitivity to seemingly irrelevant small-scale shifts in mass may mean that for a particular halo finder that breaks down mid-merger, compression may cause the break down to behave slightly differently between the compressed and uncompressed simulation (for example, leading to mass switches occurring at different snapshots). But this would not be a failure of compression: it is still a failure of the halo finder. Constructing quantitative tests of how effectively major mergers are tracked is difficult given that halo catalogues constructed on the underlying simulation cannot necessarily be treated as a reliable base truth.

Similarly, subhalo evolution may be affected by compression due to failures in subhalo finding routines. As discussed at length in Mansfield et al. (2023), it is common for the ROCKSTAR to experience catastrophic errors during the final few snapshots of a halo's life, mis-estimating its properties and potentially 'stitching' it onto spurious phase space fluctuations. Once again, in a regime where a subhalo finder is highly sensitive to noise, it is possible that slightly perturbing particles may lead to a different result.

The solution to both the above ambiguities would be to do the comparison in a subhalo analysis framework which does not experience either class of error and which makes robust estimates of how reliable a given subhalo identification is. As demonstrated in Mansfield et al. (2023), the particle-tracking subhalo finder SYMFIND and its surrounding statistical framework is a strong candidate for such a system. We cannot combine it with GUPPY at this time because GUPPY currently only works on uniform-mass cosmological simulations and SYMFIND only works on single-host, multiresolution zoom-in simulations.

Lastly, we note that one of the intermediate data products used by the SYMFIND pipeline is a set of quantized, compressed particle

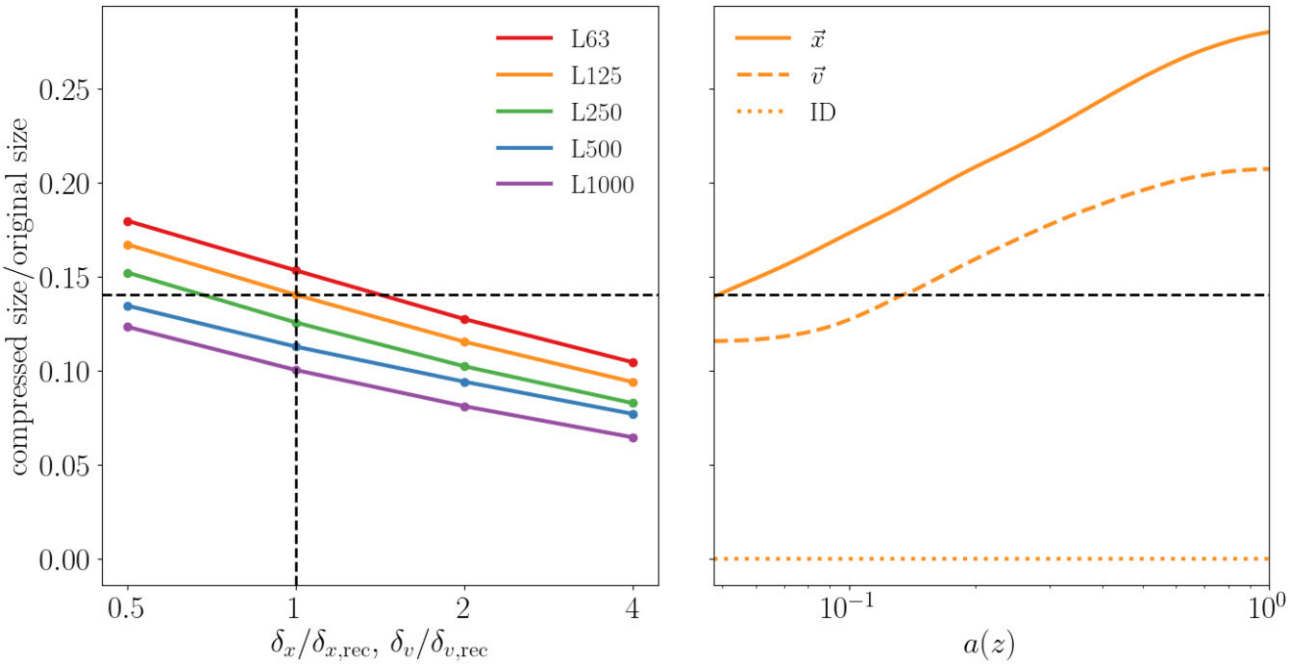

Figure 7. The size of GUPPY files relative to the original simulation files. Left: The ratio of GUPPY sizes to uncompressed sizes for various simulations in the Erebos_CBol suite as a function of accuracy. Each curve is a different simulation, ranging from the high-resolution L63 (slightly less resolution than TNG100-1-Dark) to the low-resolution L1000 (with slightly less resolution than the main DarkSkies box, ds14-a). The vertical dashed line shows the recommended resolution at which compression has essentially no impact on halo properties (see Section 3). Compression improves snapshot size by a factor of $\approx 10-7$, and decreasing accuracy only modestly improves this ratio. Right: GUPPY's compression efficiency as a function of scale factor of scale factor for different particle properties for Erebos_CBol_L125 when compressed to the recommended accuracy. The horizontal black line shows the total compression ratio GUPPY achieves across all properties and snapshots. Positions are the most difficult quantity to compress, and both positions and velocities become more difficult to compress at later times when a larger fraction of mass has collapsed into haloes.

snapshots. δ_x and δ_v for these snapshots change dynamically on a subhalo-by-subhalo basis, but in the highest resolution simulations considered in Mansfield et al. (2023), most highly disrupted subhaloes with tidal tails that have begun to phase mix with the host halo will have quantization scales are similar to the fiducial accuracy levels suggested in this work. Despite this compression, SYMFIND is able to follow heavily disrupted subhaloes to masses which are orders of magnitude lower than those achievable by ROCKSTAR without stitching errors and can follow major mergers without mass-switching errors. Thus, we find it unlikely that quantization will fundamentally hamper studies that require reliable subhalo evolution or that δ_x and δ_v would need to be insignificantly altered.

3.6 Comparison with previous work

There has been little testing of how much compression simulation snapshots can withstand without distorting halo properties. Most previous work has either focused primarily on the numerical and algorithmic properties of different compression algorithms (e.g. Tao et al. 2017) or compressed particles so accurately that there would obviously be no impact on analysis (e.g. Emberson et al. 2017). The existing tests that we are aware of consist of $\approx$ Gpc to 100 Mpc-scale images of error- and displacement-fields (Di et al. 2019) as well as power spectra and the halo mass function (Jin et al. 2020).

These are important tests and would certainly catch defects in very aggressive compression schemes, and may be entirely sufficient for analysis restricted to large scales. However, they are not sufficient to inspire confidence in general-purpose analysis which may reach small radii. As Jin et al. (2020) demonstrate, even compression accuracies which have next to no impact on the power spectrum can still be coarse enough to suppress the halo mass function at small masses. And as a comparison between Fig. 4 and Figs 5 and 6 show even accuracies that have no impact on the more stringent subhalo mass function can still bias internal halo properties substantially.

Therefore, we hope that the criteria outlined in the preceding section will allow lossy compression algorithms to be used with more confidence and that these limits will be useful to future studies of compression techniques.

4 COMPRESSION RATIOS

In Fig. 7 we show how effectively GUPPY can compress simulation snapshots as a function of accuracy. The left panel shows the ratio of compressed-to-original sizes for a variety of accuracy levels. A different curve is shown for each box in the Erebos suite. The highest resolution box, Erebos_CBol_L63 has slightly less resolution than TNG100-Dark (Marinacci et al. 2018; Naiman et al. 2018; Nelson et al. 2018; Pillepich et al. 2018; Springel et al. 2018). The lowest, Erebos_CBol_L1000 has slightly less resolution than the flagship DarkSkies box, ds14-a (Skillman et al. 2014). Each point represents a data set which was roughly 3.3 TB prior to compression.

Accuracies are shown relative to our recommended position accuracy, $\delta_{x,\mathrm{rec}} = \epsilon$ and velocity accuracy, $\delta_{v,\mathrm{rec}} = V_{\mathrm{vir},100}/8$. At the recommended accuracy level, the sizes of simulations are reduced by a factor of 7 to 10, without any meaningful bias to halo properties or small-radius profiles (Section 3).

Compressed file size only has a weak dependence on accuracy. Increasing δ_x and δ_v by a factor of 2 only reduces simulations sizes

by roughly 20 per cent. While we showed in Section 3 that most analysis could safely double δ_x and δ_v above our recommended limits, the benefits of doing so are modest, so we recommend erring on the side of caution.

Compressed file sizes show a modest dependence on simulation resolution, with lower resolutions enjoying smaller file sizes. This is due to a combination of two factors. First, the ratio of ϵ to a box's mean interparticle spacing slightly increases with decreasing resolution. This is because Erebos was tuned to study the mass–concentration relation and smaller force softening scales are needed in low mass haloes due to their high concentrations (Diemer & Kravtsov 2015). However, this is a weak trend: $\epsilon/(L/N)$ in Erebos_CBol_L1000 is only twice as large as $\epsilon/(L/N)$ in Erebos_CBol_L63. The second effect is that as particle masses become larger, the ratio of m_p to the exponential cutoff in the halo mass function becomes smaller, meaning that a larger fraction of particles have not undergone shell-crossing and are therefore easier to compress.

The right panel shows how effectively different fields are compressed as a function of scale factor in the Erebos_CBol_L125 box when compressed to our recommended accuracy limits. The solid, dashed, and dotted orange curves show the position, velocity, and ID fields, respectively, and the black dashed line shows the average compression ratio across all three. As more structure collapses in the simulation, compression becomes more difficult, leading to worse compression ratios. Our recommended accuracy limits lead to velocities being compressed more coarsely than positions, which means that they are stored more efficiently. IDs are stored implicitly and therefore take up no space.

4.1 Comparison with previous work

As discussed in the introduction, many previous authors have worked on compression algorithms which are designed for *N*-body data sets. Some methods which err on the side of simple decoding methods can modestly reduce data sizes by factors of two to three (Emberson et al. 2017; Maksimova et al. 2021). Other more aggressive and complicated schemes claim factors for four (Cheng et al. 2020) to twenty (Di et al. 2019).

It is not easy to compare methods apples-to-apples across these studies, as many different error schemes are used, and the compression ratios of all lossy compression algorithms depend strongly on accuracy. Many papers characterize compression errors in terms of $\mathrm{PSNR} = 20\log_{10}(\sqrt{\langle|x_{\rm true} - x_{\rm compressed}|^2\rangle})$, however as Jin et al. (2020) point out, different schemes with the same PSNR can have very different impacts on analysis targets. Additionally, some existing algorithms either rely on earlier snapshots to work (see Section 2.3.1) or use a relative error scheme to encode velocities (see Section 2.2.1). As we argue in the aforementioned sections, while there are certainly specific types of analysis for which either or both methods are acceptable (and even optimal), they would not be appropriate for, say, allowing users with very limited computing resources access to individual snapshots. Lastly, the compression ratio one achieves is a strong function of the fraction of particles which have collapsed into haloes, meaning that comparisons need to be performed at a fixed particle mass and redshift range.

Therefore, we do not believe that it is possible to directly compare compression ratios between these different algorithms using only published data, other than to say that many of these algorithms will presumably continue to give ≈ order-of-magnitude compression ratios when run at the same accuracy limits and make no claims that GUPPY outperforms the compression ratios of these algorithms. A comparison would require a detailed cross-comparison on the same data with the same accuracy limits. We defer this to future work.

5 DISCUSSION

Currently, the most important challenge in the field of cosmological data compression is not the development of new algorithms, but in increasing community adoption of these techniques. There currently exist numerous techniques which are capable of reducing the size of simulation snapshots by ≈ an order of magnitude (both GUPPY, and earlier works such at Li et al. 2018; Di et al. 2019), and the work in Section 3 has established clear, conservative limits on how accurately particles need to be stored. Given that the overwhelming majority of cosmological data is currently stored in an uncompressed form, the top priority of the field should be making it easier for researchers to compress their data.

In the authors' view, the most important steps forward are to integrate compression codes directly into popular cosmology software, and to provide user-friendly libraries which can read and write this data across a range of languages. GUPPY libraries are available in C, PYTHON, and GO, and as part of this paper, we have integrated it into the ROCKSTAR halo finder. This type of integration is not always trivial: in the case of ROCKSTAR, we had to alter ROCKSTAR's internal bounds-calculation and bounds-checking code (see Appendix A), and ROCKSTAR's paralellization model was incompatible with the initial multithreading scheme used by GUPPY. Both issues required substantial testing to identify. Because of this, this type of integration work could act as a very real barrier to entry, and it is important to approach it as a serious research effort rather than something that users will be able to do trivially.

6 CONCLUSIONS

In this paper, we have presented and tested the GUPPY compression algorithm which can compress dark-matter-only cosmological simulation snapshots by roughly an order-of-magnitude. GUPPY is a 'lossy' algorithm, meaning that it introduces a small amount of strictly bounded noise into particle properties which is uncorrelated between particles. We have extensively tested the properties of this noise and have identified conservative limits, $\delta_{x,\rm rec}$ and $\delta_{v,\rm rec}$, which do not have any meaningful impact on single-snapshot dark matter halo properties. Upon publication, we will release version 1.0.0 of a code implementing this algorithm[18] which can compress Gadget-2 particle files. We have also added support for the resulting files to the popular ROCKSTAR halo finder.[19]

A typical reader will likely be most interested in Section 3 (particularly Sections 3.3, 3.4) and Figs 2–6, where we identify accuracy limits which have little-to-no impact on the internal properties of dark matter haloes. In Section 2 and Fig. 1, we cover the inner workings of GUPPY. In Section 4 and Fig. 7, we show the space savings that can be achieved by the GUPPY algorithm. Lastly, in Section 5, we discuss what we view to be the most important next steps in the field of cosmological data compression. Readers interested in integrating GUPPY into existing analysis software or data pipelines may want to review Appendix A, which covers how to perform various operations in periodic space.

[18] https://github.com/phil-mansfield/guppy
[19] https://bitbucket.org/philip-mansfield/rockstar/src/main/

ACKNOWLEDGEMENTS

PM acknowledges support by the KIPAC fellowship at Stanford University. This work was supported by the U.S. Department of Energy SLAC Contract DE-AC02-76SF00515.

The authors would like to thank Benedikt Diemer and Andrey Kravtsov for access to the Erebos simulation suite and Peter Behroozi for extensive help with integrating GUPPY into ROCKSTAR. We would also like to thank Andrey Kravtsov and Nick Gnedin for discussions which helped us to improve this work, Ben Dodge for helping us to proof-read pseudocode, and the anonymous referee for comments which help improve the quality of this work. PM would like to thank Jenny Chen for emotional support throughout the writing of this paper.

The authors made extensive use of the `NumPy` (Oliphant 2015), `SciPy` (Jones et al. 2001), and `matplotlib` (Hunter 2007) libraries throughout this paper.

DATA AVAILABILITY

The code used to create the compressed snapshots in this paper is available at https://github.com/phil-mansfield/guppy, and https://bitbucket.org/philip-mansfield/rockstar/src/main/. The data used this article will be shared upon reasonable request.

REFERENCES

Angulo R. E., Springel V., White S. D. M., Jenkins A., Baugh C. M., Frenk C. S., 2012, MNRAS, 426, 2046
Behroozi P. S., Wechsler R. H., Wu H.-Y., 2013, ApJ, 762, 109
Behroozi P. et al., 2015, MNRAS, 454, 3020
Bryan G. L., Norman M. L., 1998, ApJ, 495, 80
Butz A. R., 1971, IEEE Trans. Comput., 100, 424
Cheng S., Yu H.-R., Inman D., Liao Q., Wu Q., Lin J., 2020, preprint (arXiv:2003.03931)
Cherry E., 1953, Proc. IEE-Part III: Radio Commun. Eng., 100, 9
Cherry C., Kubba M., Pearson D., Barton M., 1963, Proc. IEEE, 51, 1507
Di S., Cappello F., 2016, in IEEE International Parallel and Distributed Processing Symposium (IPDPS). Institute of Electrical and Electronics Engineers Inc., Chicago, IL, p. 730
Di S., Tao D., Liang X., Cappello F., 2019, IEEE Trans. Parallel Distributed Syst., 30, 331
Diemer B., 2017, ApJS, 231, 5
Diemer B., Kravtsov A. V., 2015, ApJ, 799, 108
Duda J., 2013, preprint(arXiv:1311.2540)
Dvořák J., Maňák M., Váša L., 2020, J. Mol. Graph. Model., 96, 107531
Emberson J. D. et al., 2017, Res. Astron. Astrophys., 17, 085
Entacher K., 1998, ACM Trans. Model. Comput. Simul., 8, 61
Errani R., Navarro J. F., 2021, MNRAS, 505, 18
Gorski K. M., Hivon E., Banday A. J., Wandelt B. D., Hansen F. K., Reinecke M., Bartelmann M., 2005, ApJ, 622, 759
Guo Q. et al., 2011, MNRAS, 413, 101
Hahn O., Abel T., Kaehler R., 2013, MNRAS, 434, 1171
Hanisch R. J., Farris A., Greisen E. W., Pence W. D., Schlesinger B. M., Teuben P. J., Thompson R. W., Warnock A. III, 2001, A&A, 376, 359
Haramoto H., Matsumoto M., 2019, preprint (arXiv:1908.10020)
Huffman D. A., 1952, Proc. IRE, 40, 1098
Hunter J. D., 2007, Comput. Sci. Eng., 9, 90
Huwald J., Richter S., Ibrahim B., Dittrich P., 2016, J. Comput. Chem., 37, 1897
Ishiyama T. et al., 2021, MNRAS, 506, 4210
Jin S., Grosset P., Biwer C. M., Pulido J., Tian J., Tao D., Ahrens J., 2020, IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, New Orleans, LA, p. 105
Jones E. et al., 2001, SciPy: Open source scientific tools for Python. Available at: http://www.scipy.org/
Klypin A. A., Trujillo-Gomez S., Primack J., 2011, ApJ, 740, 102
Klypin A., Prada F., Comparat J., 2017, preprint(arXiv:1711.01453)
Kumar A., Zhu X., Tu Y.-C., Pandit S., 2013, in Sun C., Fang F., Zhou Z. H., Yang W., Liu Z. Y., eds, Intelligence Science and Big Data Engineering. Springer, Berlin, p. 22
Le Gall D., 1991, Commun. ACM, 34, 46
L'ecuyer P., Simard R., 2007, ACM Trans. Math. Softw., 33, 1
Lemire D., O'Neill M. E., 2019, Comput. Appl. Math., 350, 139
Li S., Di S., Liang X., Chen Z., Cappello F., 2018, in IEEE International Conference on Big Data (Big Data). IEEE, Seattle, WA, p. 428
Ludlow A. D., Schaye J., Bower R., 2019, MNRAS, 488, 3663
Maksimova N. A., Garrison L. H., Eisenstein D. J., Hadzhiyska B., Bose S., Satterthwaite T. P., 2021, MNRAS, 508, 4017
Mansfield P., Avestruz C., 2021, MNRAS, 500, 3309
Mansfield P., Kravtsov A. V., 2020, MNRAS, 493, 4763
Mansfield P., Darragh-Ford E., Wang Y., Nadler E. O., Wechsler R. H., 2023, preprint (arXiv:2308.10926)
Marais P., Kenwood J., Smith K. C., Kuttel M. M., Gain J., 2012, J. Comput. Chem., 33, 2131
Marinacci F. et al., 2018, MNRAS, 480, 5113
Marsaglia G., 2003, J. Stat. Softw., 8, 1
Meyer T., Ferrer-Costa C., Pérez A., Rueda M., Bidon-Chanal A., Luque F., Laughton C., Orozco M., 2006, J. Chem. Theory Comput., 2, 251
Nadler E. O. et al., 2023, ApJ, 945, 159
Naiman J. P. et al., 2018, MNRAS, 477, 1206
Nelson D. et al., 2018, MNRAS, 475, 624
Nelson D. et al., 2019, Comput. Astrophys. Cosmol., 6, 2
Oliphant T. E., 2015, Guide to NumPy, 2nd edn. CreateSpace Independent Publishing Platform, USA
Omeltchenko A., Campbell T. J., Kalia R. K., Liu X., Nakano A., Vashishta P., 2000, Comput. Phys. Commun., 131, 78
Pence W. D., White R. L., Seaman R., 2010, PASP, 122, 1065
Pillepich A. et al., 2018, MNRAS, 475, 648
Sayood K., 2006, Introduction to Data Compression, 3rd edn. Elsevier, Amsterdam
Schaller M. e. a., 2018, Astrophysics Source Code Library, record ascl:1805.020
Skillman S. W., Warren M. S., Turk M. J., Wechsler R. H., Holz D. E., Sutter P. M., 2014, preprint (arXiv:1407.2600)
Sousbie T., Colombi S., 2016, J. Comput. Phys., 321, 644
Springel V., 2005a, User guide for GADGET-2. Max-Plank-Institute for Astrophysics, Garching, Germany
Springel V., 2005b, MNRAS, 364, 1105
Springel V. et al., 2005, Nature, 435, 629
Springel V. et al., 2008, MNRAS, 391, 1685
Springel V. et al., 2018, MNRAS, 475, 676
Springel V., Pakmor R., Zier O., Reinecke M., 2021, MNRAS, 506, 2871
Srisawat C. et al., 2013, MNRAS, 436, 150
Sterne J., 2012, MP3: The Meaning of a Format. Duke Univ. Press, Durham
Tao D., Di S., Chen Z., Cappello F., 2017, in IEEE International Conference on Big Data (Big Data). IEEE, Boston, p. 486
Vigna S., 2017, J. Computat. Appl. Math., 315, 175
Villaescusa-Navarro F. et al., 2021, ApJ, 915, 71
Wallace G. K., 1992, IEEE Trans. Consumer Electron., 38, xviii
Wang K., Mansfield P., Anbajagane D., Avestruz C., 2023, preprint(arXiv:2311.08664)

APPENDIX A: MEASURING THE BOUNDING BOXES AROUND COLLECTIONS OF POINTS IN PERIODIC SPACE

As discussed in Section 2.5, many applications call for measuring bounding boxes around collections of points in periodic space and for detecting whether bounding boxes overlap with one another. For example, load-balancing in ROCKSTAR is performed by measuring bounding boxes around the particles associated with every reader process and constructing bounding boxes for the regions of space

associated with each writer process, then sending particles from a reader to a writer when their boxes overlap with one another.

We have noticed that constructing bounding boxes in periodic conditions is a frequent point of confusion for students, and that many public codes use bounding box algorithms that are only correct in the special case where bounding boxes are smaller than $L/2$ in each dimension, where L is the width of the simulation. So in this Appendix, we review how to construct bounding boxes in periodic boundary conditions and how to test for overlap.

First, we note that each dimension of a bounding box is independent of the others, meaning that computing a bounding box reduces to computing the 1D bounding interval of the points along each dimension and that detecting overlap between two 3D bounding boxes reduces to detecting that they are simultaneously overlapping in all three dimensions. Because of this, we will focus on the bounding interval associated with a 1D set of points, x_i on the periodic range $[0,\ L)$.

Without periodic boundary conditions, the simplest way to define a bounding interval is with its minimum and maximum values, $x_{\rm min}$ and $x_{\rm max}$. This no longer works with periodicity, as a small patch of points that spans the edge of the range could have an interval as large as $x_{\rm min} = 0$, and $x_{\rm max} = L$. In periodic conditions, it is better to define intervals as a point $x_{\rm min}$ and a width, w, and to then define the bounding interval as the range with the smallest w which contains all the points.

Under this definition, finding the bounding interval reduces to finding the largest gap, $g = x_{i+1} - x_i$, between consecutive points. For the purposes of finding the bounding interval, x_N and x_1 are consecutive and their gap is $g = L - x_N + x_1$. Once the largest gap is found to be between x_i and x_{i+1}, the bounding range of the points is given by $x_{\rm min} = x_{i+1}$ and $w = L - g$.

One could identify the gaps by looping over sorted x_i values, but there is a faster approach that relies on 'bucket-sorting' the points.[20] Qualitatively, this algorithm is an approximate solution that breaks $[0, L)$ into a set of uniform width 'buckets' which each track of the left-most and right-most point assigned to that bucket. Gaps are found between the left-most and right-most particles in neighbouring buckets, potentially skipping over empty buckets. This procedure will always produce a valid bounding range and will give the smallest possible bounding range if said smallest range is smaller than 1-the size of a single bucket. For a sufficiently large number of buckets, this approximation only means that some particle distributions which span nearly all of $[0, L)$ may not have optimally small bounding boxes, but this will barely change the number of intersection checks passed by this range.

First, construct two arrays of some length M, $\min_j$ and $\max_j$. Each element of these arrays corresponds to a consecutive, length-L/M segment the full periodic range. Calculate the maximum and minimum of the x_i within each segment and store these values in the associated arrays. If a bucket is empty, one can either store a sentinel value (e.g. NaN) in that bucket or keep trackof which buckets are empty with a second array. Store sentinel values, such as -1 or NaN, in the array elements that correspond to empty segments. To find the largest gap, look at every non-sentinel maximum, $\max_j$, and pair it with a non-sentinel minimum, $\min_k$, for the smallest k that is larger than j. As a special case to handle periodicity, pair the last non-sentinel maximum with the first non-sentinel minimum. The largest gap is the maximum of $\min_k - \max_j$. This procedure will give the smallest possible bounding range if $g >= L/M$. Setting $M = 1024$ is good enough for our purposes. Pseudocode for this algorithm is given in Algorithm 1.

[20]While a version of this algorithm appears in many 'coding interview practice' compilations, we were unable to identify its original author. The earliest reference we could find was in a set of 2011 lecture notes by the late Godfried Toussaint, but it is unclear to us whether he created the algorithm. At the time of publication, these notes are hosted at http://cgm.cs.mcgill.ca/~godfried/teaching/dm-reading-assignments/Maximum-Gap-Problem.pdf

Algorithm 1 An algorithm for finding the smallest range – with starting point $x_{\rm min}$ and width w – that contains all the points, x, in a periodic range $[0, L)$. Assumes that the $[0, L)$ range has been broken up into M equal-size 'buckets'.

```
Min (array) ← minimum x in each bucket
Max (array) ← maximum x in each bucket
Start ← index of first non-empty bucket
End ← index of last non-empty bucket
g ← L + Min[Start] - Max[End]
x_min ← Min[Start]
j ← Start
while j < M do
    k ← next non-empty bucket after j
    g' ← Min[k] - Max[j]
    if g' > g then
        g ← g'
        x_min ← Min[k]
    end if
    j ← k
end while
w ← L − g
```

Two bounding ranges, $(x_{\rm min,1},\ w_1)$ and $(x_{\rm min,2},\ w_2)$ can only overlap if either $x_{\rm min,1}$ is contained within range 2, or $x_{\rm min,2}$ is contained within range 1. To check whether a point, x, is within the bounding range $(x_{\rm min}\ w)$, first check whether $x > x_{\rm min}$. If so, the range contains x if and only if $x < x_{\rm min} + w$. Otherwise, the range contains x if and only if $x_{\rm min} + w - L > x$. Pseudocode for this algorithm is given in Algorithm 2.

Algorithm 2 An algorithm for computing Overlap, a boolean which is true if two periodic ranges characterised by $(x_{\rm min,1},\ w_1)$ and $(x_{\rm min,2},\ w_2)$ overlap within a periodic range, $[0, L)$, and false otherwise.

```
procedure ContainsPeriodic(Start, End, x)
    if End <L then
        return Start ≤ x and End >x
    else
        return End - L > x or Start ≤ x
    end if
end procedure
Overlap_1 ← ContainsPeriodic(x_min,1, x_min,1 + w_1, x_min,2)
Overlap_2 ← ContainsPeriodic(x_min,2, x_min,2 + w_2, x_min,1)
Overlap ← Overlap_1 or Overlap_2
```

This paper has been typeset from a TEX/LATEX file prepared by the author.