IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 30, 2021, accepted May 14, 2021, date of publication May 28, 2021, date of current version June 9, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3084838

Effect of d-Dimensional Re-orderings on Lossless Compression of Radio-Astronomy and Digital Elevation Data

CONRAD J. HAUPT[1], EKOW J. OTOO[1,2], (Member, IEEE), AND LING CHENG[1], (Senior Member, IEEE)

[1]School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg 2000, South Africa
[2]Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

Corresponding author: Ling Cheng (ling.cheng@wits.ac.za)

This work was supported by the South Africa's National Research Foundation under Grant 129311 and Grant 132651.

ABSTRACT For multidimensional data, Space-Filling Curves (SFCs) have been used to improve the execution time of spatial data queries. However, their effect on compression, when used to reorder the uncompressed values, is known to a lesser extent. We investigate the impact of three SFCs on Shuttle Radar Topographic Mission (SRTM) elevation data and Square-Kilometre Array telescope (SKA) radio-astronomy data: two types of datasets to which SFCs have not been extensively applied, within a compression context. This work contributes to the understanding of how such reorderings impact compression performance and affect different compression schemes and preprocessing techniques through their use. We show empirical results from combining eight common compression schemes, the Z-Order, Gray-Code, and Hilbert space-filling curves, and the bitwise preprocessing technique BitShuffle. The Hilbert Curve consistently outperforms the other orderings for the SRTM dataset though the mapping implementation incurs a significant speed penalty. However, the Z-Order and Gray-Code Curves are best for the SKA dataset. Through an analysis of the dataset autocorrelations, file-entropies, and block-entropies; we show that the SKA dataset's dimensional bias is not exploited as much by the Hilbert Curve compared to the Z-Order and Gray-Code Curves. However, the Hilbert Curve is the most appropriate for the SRTM dataset as it can be modelled as isotropic and has a significantly higher level of local autocorrelation. BitShuffle is necessary to practically compress the SKA data, but does contribute to the compression performance of the SRTM dataset. These curves and BitShuffle are advantageous in reducing block-entropy values for such datasets.

INDEX TERMS Data compression, computers and information processing, digital elevation models, radio astronomy, space-filling curves, multidimensional data.

I. INTRODUCTION

Space-Filling Curves are mappings between the one-dimensional space and the d-dimensional space, and are used to improve query times of spatial data-structures by reordering and indexing the underlying values, preserving some spatial locality. Their d-dimensional orderings create curves which wrap around themselves and traverse local subregions, clustering nearby points together. This property results in some neighbouring d-dimensional values being closer in the one-dimensional space than if a standard row-major or raster scan was used; the extent to which is dependent on the type of data and SFC applied. For this reason, they have applications in areas where spatial properties must be preserved or modified, such as improving indexing performance of relational and spatial databases [1]–[3]. They have also been used to reduce I/O access times by reducing the number of disk seeks necessary for range queries [1] or improving disk address prediction from disk indices [4]. SFCs have shown promise in large point-data management software, for both data compression and access [5], [6]. Given their locality preservation properties, SFCs are effective at improving k nearest-neighbour queries [7]–[10]. As SFCs map between d and one dimensions, they are also applicable for Machine Learning (ML) scenarios where data must be mapped into an alternative form appropriate for a given ML model, in some

The associate editor coordinating the review of this manuscript and approving it for publication was Xi Peng.

cases with an improvement to the overall accuracy of said model. Instances of this in the literature include the classification of schizophrenia and normal patients' 3D fMRI scans [11], gesture recognition of Surface Electromyography (sEMG) signals represented as images [12], [13], and malware detection and classification using Support Vector Machines [14]. Within the context of low-level computing, the curves have been used in the creation of cache-oblivious loop structures that preserve the locality of data held within the cache [15]. Closer to the topic of compression, some SFCs have shown application in bandwidth reduction of signals based on their digital samples [16].

Many SFCs exist, though the literature typically focuses on a select few popular types and their variants [15], [17], [18]. The most popular three are the Peano [19], Z-Order [20], [21], and Hilbert [22] Curves. However, many variants of these have been discussed and investigated in the literature. In this paper we focus on three SFCs and Row-Major Order, which we treat as a reference SFC called the Raster Curve or Raster Scan. Metrics and measures have been defined to quantify the level of clustering or locality preservation but only within specific contexts and purposes such as nearest-neighbour searches [7], [8], spatial tree data-structure optimization [23], and hard-disk drive I/O performance [1], [24], [25].

Given their ability to map between multiple dimensions, these curves have also been applied to the compression of multidimensional data, with varying results. In 1986, Lempel and Ziv [26] showed that the compressibility of an image is lower-bounded by the same image when ordered using the Hilbert Curve [22]. They describe this behaviour using their encoder published in 1978, commonly referred to as LZ78 [27], [28]. However, their conclusion is not universal as it is dependent on the type of encoder defined in their work, one that is derived from LZ78. There are a few published cases where the use of SFCs are used to effectively compress data, and in some cases outperform the standard Raster Scan. Pinciroli *et al.* use the Hilbert Curve to compress angiocardiographic images to a compression ratio of approximately $\mathcal{R}_c = 6$, though lossy in nature [29]. Another application, also to medical images, is the work by Liang *et al.*, where Run-Length Encoding (RLE) [30], LZW [31], LZ77 [27], and Huffman Encoding (HFF) [32] are applied to CT Scan images with differential coding and reordering using the Hilbert Curve [33]. They achieve a maximum compression ratio of 3.42 utilizing LZW and HFF, with differential coding and Hilbert Curve reordering as preprocessing steps.

An application of SFCs to the compression of colour images is shown by Abdollahi *et al.* [34] where they combine Move-to-Front Encoding with various SFCs as their compression technique. They quantify the effect on compressibility, assuming an entropy-encoder such as HFF, using the entropy-ratio instead of $\mathcal{R}_c$: the ratio between the resulting image and input image's entropies. For all images they investigated, Abdollahi *et al.* achieved the lowest entropy-ratios when the Hilbert Curve was applied. Provine and Rangayyan compress greyscale images mapped with the Hilbert and Raster Curves using HFF, Arithmetic Coding, Predictive Coding, and LZW [35]. The highest compression ratios they achieve, of approximately $\mathcal{R}_c = 5$, are obtained with HFF combined with differential coding, a form of predictive coding.

In some cases, bespoke SFCs have been designed to assist with the compression of data. Ouni, Lassoued, and Abid develop a *gradient-based* SFC (GSFC) for the compression of two-dimensional images [36], [37]. When used in conjunction with the Graphics Interchange Format (GIF) [38], which uses LZW, their *vote* variant of the GSFC achieves better compression than with the Raster or Hilbert Curves [36]. However, with the PNG format [39], which uses DEFLATE [40] (a combination of LZ77 and HFF), all SFCs are approximately the same: the Hilbert Curve and GSFC achieve similar compression ratios [36]. When compared to other compression schemes, the lossless JPEG compressor JPEG-LS nearly always achieves better compression than GIF, PNG, and their Differential Pulse Code Modulation (DPCM) [41], GSFC, and HFF based technique [37]. Nevertheless, their GSFC-based compression technique achieves compression ratios just above those for JPEG-LS, indicating that further work may result in superior performance.

Scarmana and McDougall apply SFCs to the compression of Digital Terrain Models (DTMs), with results showing the fastest and best compression when the Hilbert Curve is used with DPCM [42]. They compare the compression performance of the Hilbert Curve and DPCM to LZW, JPEG2000, JPEG-LS, and DEFLATE with the Raster Scan. However, they do not compare the results to DPCM with any other orderings or SFCs. Furthermore, they only investigate three DTM files.

Even though Pinciroli *et al.* [29], Liang *et al.* [33], Abdollahi *et al.* [34], and Scarmana and McDougall [42] achieve larger compression ratios and lower entropies when utilizing the Hilbert Curve compared to the standard Raster Scan, they do not compare SFCs to the Raster Scan within the same compression schemes. Provine and Rangayyan do compare two SFCs, but one is the *base-line* Raster Curve [35]. Ouni, Lassoued, and Abid do compare the Raster Scan with two SFCs, but not all combinations of SFCs and compression schemes are explored [36], [37]. Additionally, literature covering the theoretical performance of SFCs, within the context of data compression, differ in their conclusions on the benefit of using SFCs for d-dimensional data reordering. Quin and Yanagisawa show analytically that for images made of artificial random ellipses, the use of SFC-reorderings — before a subsequent RLE compression step — does not improve the compressibility of the data [43].

They come to this conclusion by measuring the length of runs when using RLE, with longer lengths being preferable. For generic images and lossless context-based compression techniques, Memon, Neuhoff, and Shende calculate the theoretical predication gains when utilizing different ordering

techniques and conclude that there is no significant prediction gain when utilizing the Hilbert Curve over the Raster Scan [44]. They conjecture that the Raster Scan may result in better compression performance over SFCs because common compression schemes are either explicitly or implicitly designed for row-major order structured data. Whether focusing on empirical results or theoretical results, it is easily argued that existing literature does not contain the full picture on compression performance with SFC-based reorderings.

For compression of DEM data without the use of SFCs, Rane and Sapiro achieve an average compression ratio of $\mathcal{R}_c = 11.7$ with JPEG-LS [45]. The DEM data they use contains 1201×1201 16-bit integer pixels. They find that the bit-level redundancies in DEM integer data can be exploited by compression techniques, specifically JPEG-LS, if the elevation values are restructured. For example, they show that treating a 24 bit integer as two separate two and one byte integers results in better compression when using JPEG-LS; owing to the higher redundancy of the most significant two bytes.

Compression of astronomy data is dependent on the data type used, as is also the case with DEM data. Masui *et al.* develop a bit-level reordering scheme to compress radio-astronomy data [46]. Their final compression scheme BitShuffle restructures values into groups of bits by their underlying significance, resulting in higher compression ratios with compression schemes that operate on multi-byte symbols. Though they refer to BitShuffle as the preprocessing step and the subsequently applied LZ4 [47] compression scheme, it is more appropriate at times to refer to only the preprocessing step as BitShuffle [48]. Though they apply the technique to integer data, it is also applicable to floating-point data. Zheng *et al.* implement a lossless compression technique for integer astronomy data also by reformatting low-level bit representations of values [49]. Instead of grouping bits by their significance, an *effective bit-width* is used alongside DEFLATE to achieve better compression than the combination of BitShuffle and LZ4 ($\mathcal{R}_c = 3.36$ vs $\mathcal{R}_c = 2.49$) on an integer dataset from the Five Hundred Meter Aperture Spherical Radio Telescope (FAST) [48].

The work covered here, in this paper, contributes to the overall picture of compression under SFC reorderings by comparing the compression performance for two datasets of differing sources with four reorderings (including the Raster Scan) and eight common compression schemes. Though other datasets have been compressed using these curves under similar scenarios, the core of the contribution is in the usage of datasets that have not been extensively covered before. A further contribution is in the analysis of the underlying entropy-structure of the data-files which shows how the SFCs and BitShuffle affect the compressibility of the two distinct types of data used. Coupled with statistical metrics for the two datasets, this analysis expands on the knowledge of how and to what extent SFCs can be used to exploit locality within multidimensional data. We show that SFCs, as indicated by some aforementioned literature, provide a limited general improvement to compression ratios, with slight to severe reductions in speed. The impact of SFC reorderings is dependent more on the compression scheme and dataset than the specific curve applied.

The compression schemes investigated are BZIP2 [50], GZIP [40], [51] (DEFLATE), HFF [32], LZ4 [47], LZ77 [27], LZO [52], LZW [31] (a variant of LZ78 [28]), and Run-Length Encoding (RLE) [30]. BitShuffle is employed to allow further compression of both datasets and identify the impact of SFC reorderings on the least-significant binary-components of the underlying data representations [46]. BitShuffle is necessary to effectively compress one of the two datasets shown here, as the floating point data type used is not amenable to some byte-level compression techniques employed. More on the compression schemes and BitShuffle is discussed in Section II-C.

The two datasets used here are 7.72 GB of two-dimensional Digital Elevation Data from the Shuttle Radio Topographic Mission (SRTM) and 7.28 GB of three-dimensional Radio-Astronomy Correlator Data from the Square Kilometre Array (SKA) Project. These were chosen as the former expands on the work done by Rane and Sapiro [45] and Scarmana and McDougall [42]. The latter adds knowledge on the effect of SFCs to compression of a dataset to which SFCs have not yet been applied: floating-point radio-astronomy data. Further discussion of the datasets, as well as an analysis of their properties, is given in Section II-B. Further background on the compression schemes chosen and the SFCs investigated are covered and discussed in Section II. The methodology and implementation details by which the results are obtained are given in Section III. The analysis in this paper incorporates metrics used from the aforementioned literature to evaluate the impact of SFC reordering on the compression performance: compression ratios, compression and decompression speeds, file-entropy, and block-entropy. The results and analysis are given in Section IV with a final discussion in Section V.

II. BACKGROUND

The following subsections cover the technical background and explanation of the SFCs used, properties and structures of the two datasets, and a breakdown of the eight compression schemes applied. Technical aspects of the datasets and compression schemes are also given, with an explanation of the BitShuffle variant used in this work.

A. SPACE-FILLING CURVES

The first Space-Filling Curve (SFC) was discovered by G. Peano [19] in 1890 after G. Cantor proved that the cardinality of a d-dimensional unit-hypercube is the same as the unit-interval: $|\mathcal{I}| = |\mathcal{I}^d|$. Cantor's result shows that a continuous curve can *fill* a hypercube in its entirety using a surjective mapping [53], [54]. D. Hilbert [22] later found a second SFC, now referred to as the Hilbert Curve, which is more common in literature on the topic of SFCs. H. L. Lebesgue discovered the Z-Order curve by interleaving the bits of d integer

coordinate values resulting an observable zigzag pattern [20]. It was popularized by Morton in his work applying it to geodetic databases [21] and has been referred to as the Lebesgue Curve and the Morton Curve. These curves wrap around themselves and their local contexts before traversing into further regions of the d-dimensional space. This property implies that d-dimensional neighbouring points are likely to be *close* in the one-dimensional SFC-mapped space. Depending on the SFC used, the distance to a d-dimensional neighbour in the one-dimensional space may be less than with the Raster Scan on average or in the worst case [55]. This is not always the case as each curve has its own characteristics and structure. Significant research exists on evaluating which SFCs are best at this *locality preservation* process [8], [25], [56]. Of the three sources cited, the Hilbert Curve generally achieves the *best* locality preservation of the three SFCs utilized in this paper. Though different curves achieve the best metric in each paper, only the Z-Order, Gray-Code, and Hilbert Curves are covered in this paper to limit the scope.

In this paper, we refer to it as the Z-Order Curve. The Gray-Code Curve is created by treating the Z-Order one-dimensional index as a Gray Code, converting it to a Gray Code index, and then conducting the Z-Order Curve mapping on the resulting value [57]. The Raster Scan is technically an SFC; however, as it is used as the ordering for most data on disk and in memory, it is a trivial example and is instead used as the reference curve in this and other research. The Peano curve is not utilized in this work as its extents are not the same as the Z-Order, Gray-Code, and Hilbert Curves — and therefore cannot be applied to the same data without modification. All SFCs discussed here are illustrated in Fig. 1 with the following subsection covering the general definition of SFCs.

To understand the process by which indices are mapped to d-dimensional points, the SFC mapping and constraints must be defined. A given SFC $\mathcal{S}$ is defined as a surjective function from the unit-interval to the unit-hypercube. For computational purposes, we must take a finite solution of $\mathcal{S}$ to apply it to data, giving a bijective mapping. The following definition describes an SFC appropriate for computation.

Let $\mathbb{M}$ be an arbitrary finite subset of the set of natural numbers $\mathbb{N}_0$, with d-dimensional variants $\mathbb{M}^d$ and $\mathbb{N}_0^d$ respectively. We define a finite SFC mapping $\mathcal{S}$ as

$$\mathcal{S} : \mathbb{M}_x \mapsto \mathbb{M}_K^d, \tag{1}$$

where $\mathbb{M}_x$ is the set of all one-dimensional indices $\{x_i, x_j, \ldots\}$ and $\mathbb{M}_K^d$ is the set of all d-dimensional points $\{K_i, K_j, \ldots\}$. The cardinalities of the two sets are set as $|\mathbb{M}_x| = |\mathbb{M}_K^d|$ by definition. This is also the case for the surjective *true* SFCs. As all SFCs covered in this paper are constrained to a hypercube in d-dimensions, we define the size of these sets N_T, for an SFC with sidelength N, as follows:

$$|\mathbb{M}_x| = N_T = N^d = |\mathbb{M}_K^d|. \tag{2}$$

Though non-hypercubic curves exist, they are not covered in this paper. Therefore, the SFC mapping $\mathcal{S}$ transforms the index $0 \leq x_i \leq N_T - 1$ to a point $K_i = (\kappa_0, \kappa_1, \ldots, \kappa_{d-1})$ where $0 \leq \kappa_i \leq N - 1$. The SFC index is sometimes referred to as an SFC distance though this is only geometrically applicable in the one-dimensional SFC space; however, the SFC index and distances have the same values. There are four SFCs utilized in this paper: the Raster Curve/Scan $\mathcal{R}$, Hilbert Curve $\mathcal{H}$, Z-Order Curve $\mathcal{Z}$, and Gray-Code Curve $\mathcal{G}$. All of these except the Raster Curve are recursive in their definitions, as can be seen in Fig. 1, though not all SFCs need to be. Recursive SFCs (RSFCs) are defined by dividing the d-dimensional space into B subregions which are equivalent to the same SFC at a lower depth-of-recursion m. The number of regions (B) is called the base-of-recursion and is dependent on the number of dimensions d. The depth-of-recursion m is also referred to as the SFC order, allowing us to refer to a specific finite SFC as an mth order d-dimensional SFC. Of the four RSFCs in Fig. 1, the Peano Curve is the only one with $B = 3^d$, with the rest having $B = 2^d$. In fact, the Hilbert Curve is strongly related to the Z-Order and Gray-Code Curves with some mapping algorithms utilizing them as individual algorithmic steps [58]. Given the difference in B for the four

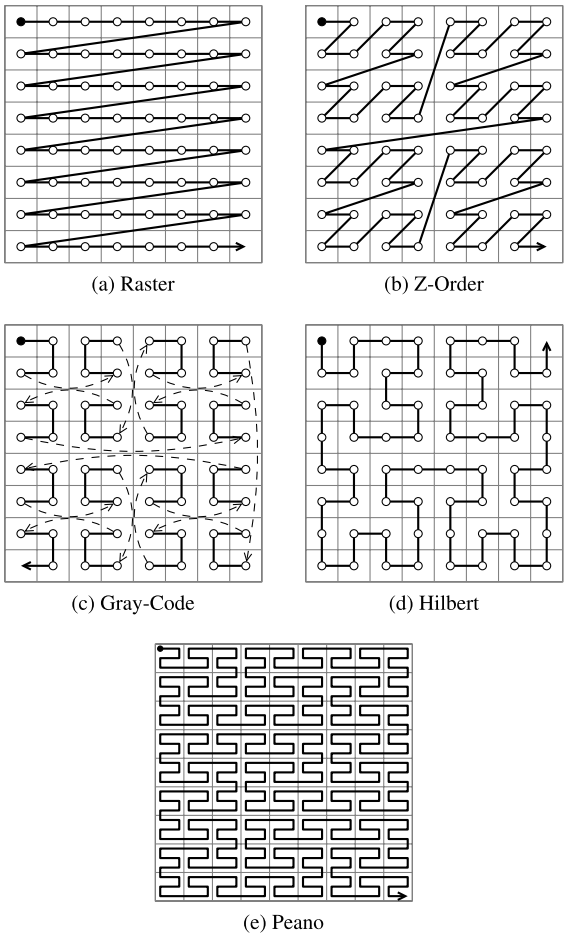

FIGURE 1. 2D representation of the types of Space-Filling Curves investigated and the Peano Curve. The markers represent points in the SFC-space with the solid marker indicating the first point along the SFC. The arrows denote the direction of traversal along the curves.

RSFCs, the Peano Curve cannot fill the same region as the other three SFCs unless modified or their *true* surjective variants are taken. The Peano Curve is not utilized here to ensure the same file extents are compressed for each configuration. Given the hypercube constraint, we can define the sidelength N of our RSFCs (Z-Order, Gray-Code, and Hilbert Curves) as

$$N = B^{\frac{m}{d}} = 2^m, \tag{3}$$

implying that $|\mathbb{M}_K^d| = |\mathbb{M}_x| = 2^{md}$. The resulting SFC mapping, given the constraints, maps the set of 2^{md} one-dimensional positive-integer indices to the set of d-dimensional coordinates. The final SFC mappings are shown below.

$$\mathcal{S}\colon \mathbb{M}_x \mapsto \mathbb{M}_K^d \tag{4}$$

$$\mathcal{S}^{-1}\colon \mathbb{M}_K^d \mapsto \mathbb{M}_x \tag{5}$$

For a given index x and its corresponding d-dimensional point K, the functions that map between them are given as

$$\mathcal{S}(x) = K = (\kappa_0, \ldots, \kappa_{d-1}) \tag{6}$$

and

$$\mathcal{S}^{-1}(K) = x, \tag{7}$$

The constraints on x and κ, for a given SFC, are

$$0 \leq \kappa_i < B^{\frac{m}{d}} \tag{8}$$

and

$$0 \leq x < B^m \tag{9}$$

Given that a dataset $\mathbb{D}$ is most likely stored in a one-dimensional representation, it must first be transformed into the d-dimensional space for the subsequent SFC mapping $\mathcal{S}$ to be applied. As most multidimensional data are in either row or column-major order, the Raster Scan SFC $\mathcal{R}$ is used: $\mathcal{R}(\mathbb{D}) = \mathbb{D}^d$. By reordering a given dataset $\mathbb{D}^d$ using an SFC, $\mathcal{S}^{-1}\left(\mathbb{D}^d\right)$, the locality of the points is somewhat preserved: i.e. neighbouring points in the d-dimensional space are *close* in the one-dimensional space. Note that the inverse of the SFC mapping is taken as the reordering must output a one-dimensional array. The full process, from row-major order to the SFC mapped array is defined as $\mathcal{S}^{-1}\left(\mathbb{D}^d\right) = \mathcal{S}^{-1}\left(\mathcal{R}(\mathbb{D})\right)$.

The work in this paper evaluates how effectively this reordering improves the compression performance of some compression techniques. The initial hypothesis was, given the evidence in the literature, that a compression scheme $\mathcal{C}$ would reduce the total size of a dataset more if an SFC mapping is used. The hypothesis is shown in (10) where $|\mathbb{D}|$ denotes the total size of the dataset $\mathbb{D}$ and $\mathcal{C}(\mathbb{D})$ is the application of compression technique $\mathcal{C} \in \mathbb{C}$ to the same dataset.

$$|\mathcal{C}\left(\mathcal{S}^{-1}\left(\mathbb{D}^d\right)\right)| < |\mathcal{C}\left(\mathcal{R}^{-1}\left(\mathbb{D}^d\right)\right)| \tag{10}$$

In this paper we use the compression ratio $\mathcal{R}_c$ as shown in (11). As is discussed in Section IV, the extent to which the compression performance is improved is heavily dependent on the dataset and whether BitShuffle is applied.

$$\mathcal{R}_c = \frac{|\mathbb{D}|}{|\mathcal{C}(\mathbb{D})|} \tag{11}$$

B. DATASETS

Two separate datasets were identified as appropriate for evaluating the impact of SFC reordering on compression schemes: Digital Elevation Model (DEM) data and radio-astronomy correlator data. Their properties are summarized in Table 1. The former is a selection of files from the Shuttle Radio Topographical Mission (SRTM) sourced through the United States Geological Survey (USGS). The dataset files have a spatial resolution of 1-arc second or 30 metres per pixel [59], [60]. When Memon, Neuhoff, and Shende conclude that the Hilbert Curve does not improve prediction gains for context-based predictive lossless encoders, they do so under the assumption that the data can be modelled as an Isotropic Gaussian Random Field (IGRF) [44].

Existing literature indicates that DEM data can be simulated using IGRFs [61], [62], making the SRTM dataset a good real-world candidate for the model chosen by Memon, Neuhoff, and Shende [44]. Empirical evidence for the SRTM data following this model is given in Section II-B2. The SRTM dataset is significantly larger than the three files used by Scarmana and McDougall [42] and the hundred files used by Rane and Sapiro [45]: three 8 MB files versus one hundred 2.75 MiB files versus nine hundred and twenty 8 MiB files. Their data and the SRTM dataset have the same data type (signed 16 bit integers) though only that used by Scarmana and McDougall and the SRTM dataset have similar extents per file (2000×2000 versus 2048×2048 pixels). The DEM files used by Rane and Sapiro measure 1201×1201 pixels in size. The second dataset is sourced from the South African Radio Astronomy Observatory (SARAO) archive [63], which is the South African component of the Square-Kilometre Array (SKA) radio-astronomy project [64]. Once completed, the SKA will be the largest radio-telescope in history. Originally created in 2018, this specific SKA dataset stores electromagnetic interference measurements in an HDF5 file [65]. The SKA measures the electromagnetic power-spectrum from hundreds of dishes, positioned around both Southern Africa and Australia, which is then correlated and processed to filter out noise and amplify low-power emissions from distant sources. The resulting data are three-dimensional in nature, with components *time*, *channel*, and *correlator products* [66]. Each channel is a fixed band of the frequency spectrum while the correlator products encode the measurements' distribution throughout the physical array. The data are stored as IEEE-754 standard single-precision floating-point values [67], presenting different bit-level behaviour to the SRTM integer data. Though typically two floating-point values are stored, representing the complex power-spectrum, the specific dataset acquired has large regions where only the real power-spectrum is present.

TABLE 1. Properties of the two investigated datasets. *Data Type* names are from the *C/C++* language standards in little-endian. The file properties shown here are for the processed and prepared files acquired from the original datasets. The names and labels for each dimension in the given datasets are provided.

	Processed Datasets	
	SRTM (A)	SKA (B)
Total Size	7.72 GB	7.28 GB
File Size	8 388 617 B	8 388 617 B
Data Type	*int16*	*float32*
File Shape	2048 × 2048	128 × 128 × 128
# Files	920	868
Dim. 1	Longitude	Time
Dim. 2	Latitude	Channel (frequency)
Dim. 3	—	Correlator Products

Other datasets from the SKA and SARAO contain the full complex power-spectrum though permission could not be acquired to utilize them. For this reason, the imaginary component was removed to not bias the results presented in this paper. As is discussed in Section II-B2, the SKA dataset does not present the same locality correlation and isotropic behaviour as the SRTM files, which is one of the reasons the dataset was chosen. By comparing datasets with different data types and models, the impact of the SFCs on compression performance can be investigated in more detail.

1) DATA PREPARATION

The SRTM and SKA datasets were sourced in the GeoTIFF [68] and HDF5 [65] formats respectively. To maintain a common format from which to apply the SFCs and compression techniques, a custom binary file-format called SFCC was created to store the same values as those stored in the original dataset files. Each SFCC file contains a nine-byte header storing metadata to assist subsequent processing steps. This includes the number of bytes in the data type, number of dimensions, the sidelength of the data extents, the current SFC ordering applied, the current compression scheme applied, whether BitShuffle is applied or not, and a four-byte magic-word to identify SFCC files. Each dataset was split into standalone files which maintained the same dimensionality as the original data but with different file extents. The sidelengths were kept as powers-of-two, given the mathematical constraints on the chosen SFCs, such that files from both datasets have the same total file-size of 8.38 MB.

Each SRTM GeoTIFF file was divided into four overlapping quadrants to translate the 3601 × 3601 data extents into four 2048 × 2048 SFCC files (extents is measured in pixels or values). The SKA data were chunked into cubes of sidelength 128 with no overlap. Owing to the data types being two and four bytes for the SRTM and SKA datasets respectively, the resulting SFCC files all measure just over 8.38 MB. The final processed datasets contain 868 SKA and 920 SRTM SFCC files, where the values are stored in row-major order.

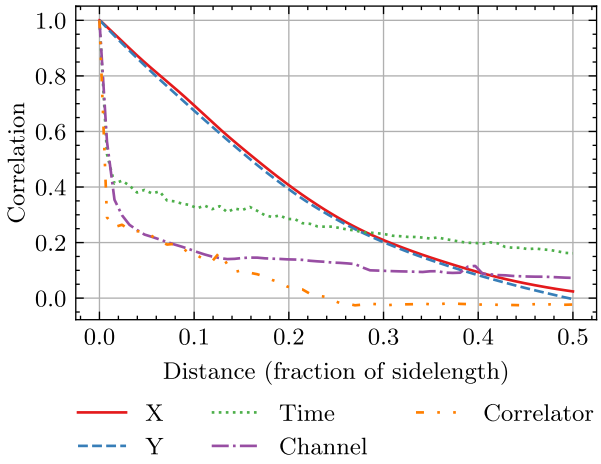

FIGURE 2. Autocorrelation for both datasets along all dimensions. The distance is measured as a fraction of the *d*-dimensional SFC-space sidelength *N*. X and Y are longitude and latitude in the SRTM data. Time, Channel, and Correlator are the three dimensions in the SKA data. Note that the sidelength for the SKA data is a 16th of that of the SRTM data: 128 versus 2048.

2) DATASET MODELS

The autocorrelation data plotted in Fig. 2 support the pre-existing assumptions that the SRTM data, being a DEM dataset, can be modelled as an Isotropic Gaussian Random Field. The *X* and *Y* plots in Fig. 2 show that elevation-values are highly correlated when they are close in the *d*-dimensional space. The similarity between *X* and *Y* indicates that the SRTM data are not biased to either dimension and can be classified as isotropic.

The SKA data show different characteristics, with a sharper reduction in the autocorrelation, more so given the smaller sidelength used (128 versus 2048). The discrepancies between the *time*, *channel*, and *correlator* dimensions indicate that the SKA data may be anisotropic, though the lack of significantly high correlation means this conclusion cannot be confirmed in its entirety. However, it is important to note that unlike the SRTM data, the SKA data reach a noticeable floor of zero correlation for the *correlator* dimension, indicating a finite range of non-zero correlation in its direction. The bias towards the *time* dimension is hypothesized to originate from EMI sources measured by the instruments, where the electromagnetic spectrum channels used are fixed and the duration of interference is sustained, creating characteristic *FFT waterfall* patterns in the data [69].

C. COMPRESSION SCHEMES

Compression schemes can be split into multiple categories, with each individual scheme potentially being comprised of multiple preprocessing steps and compression algorithms. As an example, BZIP2 is a combination of RLE, the Burrows-Wheeler Transform (BWT), Move-to-Front Encoding (MTF), and Huffman Encoding (HFF) [50], [70]. Compression algorithms are typically defined by the

TABLE 2. Composition of compression schemes and file-formats split into preprocessing steps (Preproc.) and compression algorithms. Arithmetic coding (ARITH.) and Predictive coding (PRED.) are truncated to reduce space requirements. Schemes have their own software and tools that can compress files whereas algorithms are standalone methods for compressing. The DEFLATE algorithm is indicated by ×[1]. Instances where the implementation of a given algorithm differs from the canonical definition are marked with ×[2]. Schemes[3] denote compression tools that are supported by HDF5. Algorithms and schemes investigated are underlined.

	Preproc.			Compression Algorithms						
Schemes	BWT	MTF	ST	ARITH.	HFF	JPEG	LZ77	LZW	PRED.	RLE
BZIP2[3]	×	×			×					
GZIP[3]					×[1]		×[1]			
LZ4[3]							×[2]			
LZO[3]							×[2]			
SZIP[3]			×	×						
Unix Comp.								×		
ZSTD[3]					×[1]		×[1]			
Formats										
GeoTIFF					×[2]	×		×		×[2]
IMG										×[2]
JPEG-LS									×	
PNG					×[1]		×[1]			
SRTM										
DTED										
USGS DEM										

general technique used to reduce the size of dataset. Entropy Encoding, Dictionary Compression, Arithmetic Compression, and Predictive Coding are a few examples of such general techniques. Table 2 marks which preprocessing techniques and compression algorithms make up the compression schemes and data-formats identified in the aforementioned literature and sources of the chosen datasets. HDF5 is not listed as a standalone format and instead compatible compression schemes are marked with a footnote marker.

Four standalone algorithms were chosen as good candidates for this work as they are used as components of many of the schemes and formats identified: Huffman Encoding (HFF), LZ77, LZW, and Run-Length Encoding (RLE). BZIP2 and GZIP were included as they are commonly found in popular Linux distributions [71]. LZ4 [47] and LZO [52] were chosen as they are modifications of LZ77 that are supported by HDF5, the format for the SKA dataset.

The implementations of HFF [72], LZW [72], RLE [72], and LZ77 [73] differ from other implementations as their canonical definitions do not define file-format requirements and other implementation details. For example, the Huffman encoding software used employs canonical variable-length codes, a static tree, and stores the tree in the first 256 bytes of the compressed data-block. The HFF implementation used in GZIP/DEFLATE can use either fixed or dynamic codes [40]. As is discussed in Section IV, the implementation of RLE used is somewhat *naive* as runs of 1 are not encoded differently to longer runs, thus increasing file sizes where run lengths are short and/or infrequent. Technical parameters for the compression methods used are given in Table 3.

TABLE 3. Configuration parameters for compression schemes used. Block size refers to the size of independently compressed sections of data. Symbol size is the number of bytes from the data that are treated as a standalone value for dictionary creation and encoding purposes. Some values are not listed as they are implied by the compression level of the given tool. [1]The RLE symbol size shown is for *int16* and *float32* data respectively. [2]Some values are the maximum sizes to which that the metric is allowed to expand.

Comp. Type	Block Size	Dictionary Size	Symbol Size (bytes)	Comp. Level
BZIP2	900 kB			9
GZIP			1	9
HFF			1	
LZ4	4 MB			9
LZ77		64 KiB[2]	1	
LZO	256 KiB			9
LZW		6 KiB[2]	1	
RLE			2,4 bytes[1]	

Of the eight schemes used in this research, half are provided by *external* tools within the Linux testing environment. The *internal* schemes are integrated directly into the compression testing software as they are canonical algorithms without a standard tool or program. Preliminary testing did not significantly compress the SKA dataset. To explore the lack of compressibility in these data and enhance compression performance, the BitShuffle process is implored as an extra preprocessing step for both datasets and all compression schemes.

1) BITSHUFFLE

BitShuffle is a bit-wise preprocessing technique designed to enhance the subsequent compression of radio-astronomy data, specifically for HDF5 [46]. It extends the foundational premise of shuffle, a preprocessing filter integrated into HDF5, where bytes are reordered into groups by their significance in their higher-level data type [65], [74]. BitShuffle reorders the data by bit instead of byte and, when combined with lossy precision reduction and LZ4, has been shown to achieve a compression ratio of about 3.58 on radio data [46]. In the published definition of BitShuffle, Masui *et al.* found the next best compression ratio at 3.30 with DEFLATE at a compression level of seven but with an order-of-magnitude slower speeds [46]. Of the other compression schemes they cover, the one with speeds closest to the aforementioned BitShuffle combination achieves a compression ratio around 40 % worse.

As this paper only covers lossless techniques, the variant of BitShuffle used here does not employ the precision reduction or LZ4 compression step covered by Masui *et al.* [46]. Instead, here and for the rest of this paper, BitShuffle refers to only the preprocessing bit-level restructuring. Applying

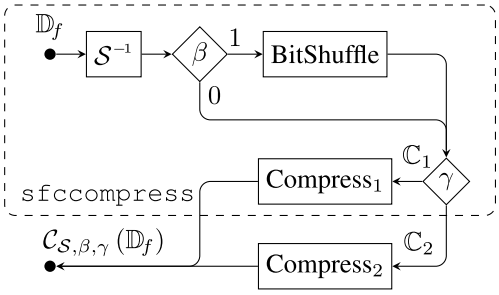

FIGURE 3. Compression process utilizing `sfccompress` and external compression tools. Internal and external compression methods/tools are labelled as $\mathbb{C}_1$ and $\mathbb{C}_2$ respectively. $\beta = 1$ when BitShuffle is applied and 0 otherwise. γ indicates which compression scheme is to be used. The compressed output for an input file $\mathbb{D}_f$ is $\mathcal{C}_{\mathcal{S},\beta,\gamma}(\mathbb{D}_f)$. Decompression is carried out in the reverse order.

BitShuffle to dataset $\mathbb{D}$,

$$\mathbb{D} = \{v^0, v^1, v^2, \ldots, v^N\},$$

where v^i has the binary structure

$$v^i = v_0^i\, v_1^i\, v_2^i\, \cdots\, v_b^i,$$

results in the BitShuffled output

$$\mathbb{D} \overset{BitShuffle}{\rightarrow} \{\bar{v}^0, \bar{v}^1, \bar{v}^2 \ldots, \bar{v}^b\},$$

where $\bar{v}^i$ has the binary structure

$$\bar{v}^i = v_i^0\, v_i^1\, v_i^2\, \cdots\, v_i^N.$$

If N is divisible by eight, then $\bar{v}^i$ is byte-aligned. The BitShuffle process does not increase the file-size. The metadata indicating that BitShuffle has been applied to an SFCC file is encoded in the most significant bit of the header data type byte.

III. METHODOLOGY

To apply the SFCs, BitShuffle, and compression schemes to the two datasets, the *C++* program `sfccompress` was developed with an accompanying Python script, `sfcc_compress.py`, for automation. Both facilitate the reordering of SFCC input data using either the Raster Scan, Z-Order, Gray-Code, or Hilbert Curves, the application of BitShuffle, and the compression of the resulting preprocessed data. BZIP2, GZIP, LZ4, and LZO — being *external* tools — are applied separately to the `sfccompress` program. The `sfcc_compress.py` script acts as an umbrella, executing the appropriate tools to apply a given configuration to a dataset file. The full compression process is illustrated in Fig. 3. The `sfccompress` program operates on SFCC files which contain raw data from the two datasets and metadata to facilitate compression, as discussed in Section II-B1.

The process is parametrized using $\mathcal{S}$, β, and γ, representing the chosen SFC reordering, whether BitShuffle is applied (1 for yes, 0 for no), and the compression scheme to use, respectively. The compressed version of an SFCC file $\mathbb{D}_f$, for a given configuration, is given as $\mathcal{C}_{\mathcal{S},\beta,\gamma}(\mathbb{D}_f)$. As an example, compressing a file with the Hilbert Curve, BitShuffle, and GZIP gives $\mathcal{C}_{\mathcal{H},1,\text{GZIP}}(\mathbb{D}_f)$. Execution of the compression and decompression processes is conducted using GNU Parallel [75] with four threads, the number available on the computing hardware used. An input list of parameters and filenames is supplied to GNU Parallel which passes them on to `sfcc_compress.py` where the appropriate tools are called. Timing is measured in milliseconds by GNU Parallel. The ordering of parameters and filenames is such that decompression jobs cannot occur before the compressed output is created in a previous job. As the process is lossless, the validity of each step was verified by reversing the process and confirming the final output matches the initial input using a SHA256 hash-sum. The full compression and decompression process was conducted over two days on a desktop computer with an Intel Core i5-2500K CPU, 16 GB of 1333 MHz DDR3 memory, a 500 GB Seagate Barracuda SSD with EXT4 partitions, the GNU C Compiler (GCC) version 9.3.0, and Ubuntu 20.04 with version 5.4.0-47 of the Linux kernel.

Storage requirements for each file is measured in bytes using the `du` Linux tool. Compression and decompression speed is calculated as megabytes-per-second (MB/s) by dividing the size of each dataset, uncompressed, by the sum of execution times for a given configuration. The uncompressed size is used for both the compression and decompression speeds. The runtime is measured as the number of milliseconds elapsed from the start of the *sfcc_compress.py* script execution to the return of an exit code. Compression ratios are calculated as the ratio of the uncompressed size and the compressed size (see (11)). Other measurements made include statistical properties of the preprocessed files using Python. File entropy is measured on a byte level, normalized between zero and one. The block-entropy of each preprocessed file is calculated with a blocksize of 16 KiB, also normalized between zero and one.

A. IMPLEMENTATION DETAILS

The Raster Scan mapping in `sfccompress` is implemented using modulo arithmetic on integer variables. For the Z-Order and Gray-Code Curves, *libmorton* is used [76]. Unfortunately, the CPU used does not support the BMI2 or AVX2 instruction sets, which *libmorton* can use to improve mapping speeds. However, this did not appear to be an issue with the given dataset sizes. The Gray-Code Curve mapping implementation uses the same *libmorton* calls but with six bitwise operations to translate between the Gray-Code index and value.

The mapping algorithm by Chenyang, Hong, and Nengchao [77] is used for the Hilbert Curve components of `sfccompress` by integrating the *libhilbert* library [78]. The SFC mapping rates from `sfccompress` are shown in Fig. 4. The Hilbert mapping is the slowest by a significant margin while the other three SFCs achieve similar mapping rates.

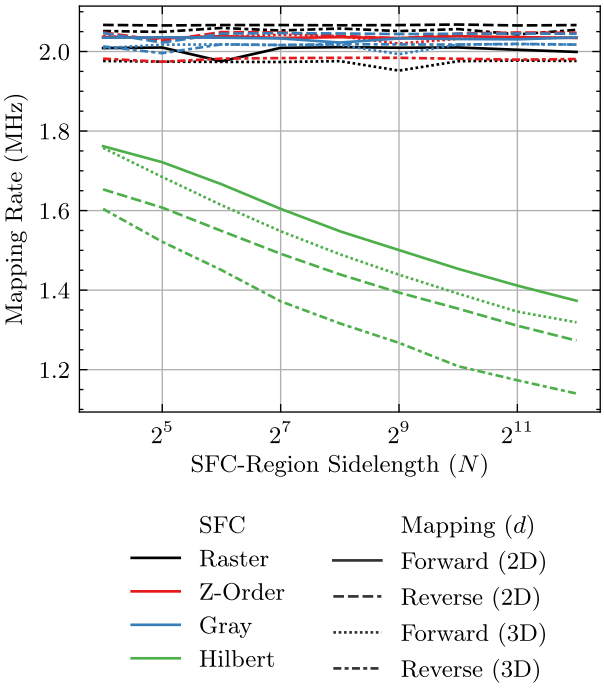

FIGURE 4. Mapping Rate for the four SFCs on the benchmarking computer. The sidelengths for the 2D SRTM dataset is 2048 (2^{11}) and the 3D SKA dataset is 128 (2^7).

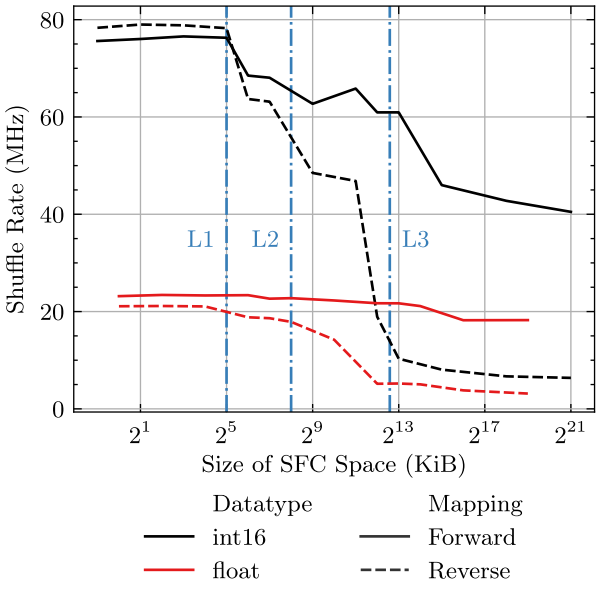

FIGURE 5. BitShuffle implementation shuffle rate vs the size of the data being mapped. All files compressed in this paper have a 2^{13} KiB sized SFC space. Note that the SRTM dataset values are 16 bit integers and the SKA dataset values are single-precision floating-point values. The L1, L2, and L3 cache sizes are shown in blue: 32 KiB, 256 KiB, and 6144 KiB respectively. The shuffle rate is measured in values per second.

Furthermore, the Raster, Z-Order, and Gray-Code mappings are not affected by the increase in sidelength. This is because the algorithm by Chenyang, Hong, and Nengchao iterates through every recursive subregion of the Hilbert mapping and is thus dependent on the SFC order m and the sidelength $N = 2^m$ [77]. However, the other three SFC mappings are not dependent on the sidelength while $N^d \leq 2^{64}$ as their mapping algorithms operate on 64-bit integer variables in the same manner irrespective of the actual value of N. For significantly larger sidelength values, more integer variables would be needed by the algorithms, incurring a small runtime penalty.

The implementation of BitShuffle only conducts a bit-level reorganization of the underlying values, as described in Section II-C1, and not the lossy precision reduction defined by Masui *et al.* [46]. This BitShuffle implementation is coded in *C++* and integrated into the `sfccompress` program. It is important to note that the BitShuffle process is dependent on the data type being used but not the number of dimensions. For the SRTM and SKA datasets, the values are 16 bit signed integers and single-precision floating-point values respectively, corresponding to two and four bytes each. For an n bit data type, blocks of n values are shuffled at a time, packing their bits into the output array based on their bit-level significance. Therefore, a single block processes $\frac{2\times n^2}{8}$ bytes at a time: 64 B for *int32* and 256 B for *float*. The performance of this process is shown in Fig. 5. The shuffle rate quantifies how many n bit input values are *shuffled* per second while the size of the SFC space is the size of a given data file, excluding all metadata ($\frac{n}{8} \times N^d$).

The sizes of the L1, L2, and L2 caches for the Intel i5 CPU used are shown in Fig. 5 in blue. The shuffle rate for both the forward and reverse mappings decreases when the SFC Space exceeds the L1 Cache size but settles at a floor after the L3 Cache size. All four cores in the i5 2500K CPU share the L3 Cache whereas there are dedicated L1 and L2 Caches for each of the four cores. However, the BitShuffle implementation is not parallel and therefore only one L1 Cache and one CPU core are used. The float data type has a shuffle rate of about a quarter of the 16 bit integer data type. Normalizing for data type size, BitShuffle on single-precision float data is still half as fast as with the smaller data type. For a given size of the SFC space, different data types result in different access patterns for the input and output arrays. Therefore, it is hypothesized that the slower shuffle rate for the float data type is primarily caused by cache misses in the BitShuffle *C++* implementation used. Secondly, the greater decrease for the reverse mapping compared to the forward mapping may also be attributed to cache misses and out-of-cache memory accesses. The memory *read* access pattern for the forward mapping is the memory *write* access pattern for the reverse mapping, and vice versa.

IV. EXPERIMENTAL RESULTS

Results are separated into three categories — compression ratios, compression and decompression speeds, and information entropy — where both datasets are covered. There are two important baseline configurations for the compression ratio and speed results. The most common in real-world applications is that without SFC reordering and without

TABLE 4. Empirical results for all combinations of compression schemes, BitShuffle, and SFC reorderings with both datasets. Measurements shown are compression ratio $\mathcal{R}_c$, compression speed, and decompression speed. The highest result for a given combination of BitShuffle β, compression scheme, and dataset is in bold and underlined. Where all values are the same, none are in bold or underlined. Note that β denotes whether BitShuffle [46] is applied ($\beta = 1$) or not ($\beta = 0$). Compression schemes shown are BZIP2 [50], GZIP (DEFLATE) [40], [51], Huffman Encoding (HFF) [32], LZ4 [47], LZ77 [27], LZO [52], LZW [31] (a variant of LZ78 [28]), and run-length encoding (RLE) [30]. All results are shown to three significant figures.

			SRTM Dataset							
			BZIP2	GZIP	HFF	LZ4	LZ77	LZO	LZW	RLE
Comp. Ratio ($\mathcal{R}_c$)	$\beta = 0$	**Raster**	**6.54**	3.52	1.44	**3.18**	**2.70**	2.96	2.39	**1.13**
		Z-Order	5.25	3.52	1.44	2.94	2.52	3.04	3.09	1.05
		Gray	5.04	3.45	1.44	2.88	2.46	3.00	3.03	1.05
		Hilbert	5.94	**3.76**	1.44	3.13	2.68	**3.21**	**3.28**	1.11
	$\beta = 1$	**Raster**	4.02	4.06	1.98	3.67	3.51	3.70	3.21	2.13
		Z-Order	4.40	4.31	**2.05**	3.97	3.99	3.97	3.42	2.41
		Gray	4.38	4.29	**2.05**	3.94	3.95	3.96	3.41	2.40
		Hilbert	**4.47**	**4.33**	2.04	**4.02**	**4.06**	**4.03**	**3.45**	**2.46**
Comp. Speed (MB/s)	$\beta = 0$	**Raster**	7.43	**6.05**	**24.7**	**6.58**	**15.6**	**3.00**	**7.07**	**40.0**
		Z-Order	7.34	3.16	17.7	5.65	11.9	1.74	6.86	29.1
		Gray	**7.48**	3.24	17.3	5.78	11.6	1.79	6.83	27.0
		Hilbert	3.56	2.09	4.99	3.02	4.63	1.36	3.45	5.49
	$\beta = 1$	**Raster**	**8.88**	4.81	**22.9**	10.8	**9.39**	2.79	**7.20**	**30.8**
		Z-Order	7.93	**5.36**	16.7	**11.0**	8.83	**3.22**	6.84	22.2
		Gray	7.63	5.11	16.2	10.4	8.90	3.11	6.89	23.0
		Hilbert	3.71	2.89	4.81	4.27	4.14	2.19	3.42	5.23
Decom. Speed (MB/s)	$\beta = 0$	**Raster**	**7.43**	**6.01**	**23.4**	**6.58**	**15.5**	**2.98**	**7.10**	**30.4**
		Z-Order	6.91	3.07	17.6	5.29	11.3	1.71	6.88	21.5
		Gray	6.51	3.04	17.0	5.17	11.2	1.73	6.70	21.1
		Hilbert	3.51	2.08	4.89	2.98	4.62	1.36	3.40	5.29
	$\beta = 1$	**Raster**	**7.58**	4.40	**20.2**	**8.94**	**9.20**	2.64	**7.19**	**26.8**
		Z-Order	6.71	**4.81**	14.6	8.92	8.66	**3.00**	6.73	21.7
		Gray	6.69	4.61	15.8	8.67	8.71	2.91	6.89	21.3
		Hilbert	3.46	2.74	4.79	3.92	4.13	2.11	3.37	5.19
			SKA Dataset							
			BZIP2	GZIP	HFF	LZ4	LZ77	LZO	LZW	RLE
Comp. Ratio ($\mathcal{R}_c$)	$\beta = 0$	**Raster**	1.06	1.10	0.941	**1.03**	**1.03**	**1.03**	0.822	0.800
		Z-Order	**1.07**	**1.12**	0.941	**1.03**	**1.03**	**1.03**	0.834	0.800
		Gray	**1.07**	**1.12**	0.941	**1.03**	**1.03**	**1.03**	0.835	0.800
		Hilbert	1.06	1.10	0.941	1.01	1.01	1.01	**0.840**	0.800
	$\beta = 1$	**Raster**	1.20	1.19	0.915	1.16	1.17	1.16	0.884	0.820
		Z-Order	**1.23**	**1.25**	**0.952**	**1.22**	**1.24**	**1.22**	**0.905**	**0.882**
		Gray	**1.23**	1.24	**0.952**	**1.22**	**1.24**	**1.22**	**0.905**	0.868
		Hilbert	1.21	1.20	0.948	1.19	1.21	1.19	0.891	0.876
Comp. Speed (MB/s)	$\beta = 0$	**Raster**	4.14	10.4	**18.7**	12.1	**2.44**	6.31	**3.21**	**35.9**
		Z-Order	4.25	11.4	15.9	14.3	2.41	**6.86**	3.16	32.5
		Gray	**4.36**	**11.5**	14.8	**14.4**	2.41	6.75	3.10	33.3
		Hilbert	3.39	6.50	7.79	7.48	2.17	4.67	2.62	10.9
	$\beta = 1$	**Raster**	**4.16**	**6.68**	**14.4**	**10.8**	2.81	**3.68**	**3.23**	**24.3**
		Z-Order	4.07	5.93	12.6	9.84	**2.86**	3.34	3.19	19.2
		Gray	4.14	5.83	12.6	9.66	2.85	3.42	3.20	21.5
		Hilbert	3.36	4.19	6.95	6.16	2.52	2.78	2.64	9.20
Decom. Speed (MB/s)	$\beta = 0$	**Raster**	4.07	10.0	**15.2**	9.80	2.41	6.03	**3.10**	20.9
		Z-Order	4.22	9.94	13.4	11.4	**2.37**	5.98	3.03	**23.2**
		Gray	**4.26**	**10.5**	12.7	**12.0**	2.35	**6.24**	3.05	19.2
		Hilbert	3.37	6.11	7.75	6.56	2.15	4.41	2.59	9.68
	$\beta = 1$	**Raster**	3.93	**5.85**	**11.8**	8.44	2.80	3.40	**3.17**	**17.9**
		Z-Order	4.08	5.81	11.1	**9.01**	**2.88**	3.31	3.14	14.4
		Gray	**4.09**	5.73	10.3	8.81	2.86	**3.41**	3.07	16.4
		Hilbert	3.28	4.04	6.54	5.72	2.51	2.71	2.57	8.75

BitShuffle — equivalent to compressing the data without any preprocessing: this is the *default* scenario. We refer to this as the unpreprocessed data as no preprocessing has been carried out on the dataset values. The second baseline is when the Raster Scan is used, as is the case with the former, but with BitShuffle applied. This scenario is mostly applicable to the SKA dataset as BitShuffle is compatible with HDF5, the format in which the data were acquired, and was designed for the type of data present in the SKA dataset. Therefore, it is likely that BitShuffle is already incorporated into other radio-astronomy datasets. However, none of the DEM file-formats support BitShuffle, and it is less likely that BitShuffle would be applied to such datasets. In each of the following subsections the key observations are identified and important comparisons raised. A discussion of the results is given in Section V. Empirical compression ratio and speed results for all configurations are given in Table 4.

A. ENTROPY ANALYSIS

The SKA dataset has high entropy, with all unpreprocessed and preprocessed files being above 0.9. The SRTM dataset has a broader range of file-entropy values, with the majority lying between 0.25 and 0.65. As the order of values does not play a role in file entropy, the SFCs do not have any effect on their own. When BitShuffle is applied, the byte values used to calculate the file entropy are changed, resulting in different file-entropies. Under this scenario, the Raster Scan results in higher entropy values for the SRTM dataset compared to all three SFCs. The Z-Order, Gray-Code, and Hilbert Curves all achieve drastically similar file-entropy values for the SRTM dataset whereas the Hilbert Curve deviates from the other two with the SKA dataset.

Applying BitShuffle to the SKA dataset, without SFC reordering, increases file-entropies by approximately 0.05. The similarity in file entropy observed between each SFC with the SRTM dataset is not entirely present in the SKA dataset, as the Hilbert Curve with BitShuffle deviates slightly to higher entropy values compared to the Z-Order and Gray-Code Curves. Given that the BitShuffle process reorders by bit and the file entropy is measured on the byte level, the deviation between the three SFCs with the SKA dataset and not with the SRTM dataset can be attributed to two factors. Firstly, the three dimensions in the SKA dataset have significantly lower autocorrelations compared to the SRTM dataset. Given that the three SFCs traverse subquadrants in different orders, the bit-level statistics of the BitShuffle values differ more with lower autocorrelation values. Secondly, the Z-Order and Gray-Code Curves traverse the least-significant dimension first whereas the Hilbert Curve traverse the most significant first (see Fig. 1).

Given that the three SFCs are recursive, it is also important to note that each subquadrant of the Hilbert Curve has a different orientation but the other two SFCs do not. Furthermore, Mokbel, Aref, and Kamel show that the Z-Order and Gray-Code Curves have dimensional bias whereas the Hilbert Curve does not [56]. The combination of dimensional-bias in the SKA data, indicated by the autocorrelation and the dimensional bias of the Z-Order and Gray-Code Curves, causes the resulting BitShuffle values to differ from the Hilbert Curve. This is not observed for the SRTM dataset as it is isotropic and thus the dimensional-bias of the curves cannot significantly influence the BitShuffle values and thus the file-entropy. However, total file entropy is not a good indicator of the underlying file structure. To analyse the impact of SFC reordering and BitShuffle further, the block-entropies for all files are plotted in Fig. 6. The similarity between all orderings without BitShuffle for file-entropies is not present in the block-entropies as each block includes different values. The blocksize is of the form 2^{dm} (for $d \in \{2, 3\}$), equivalent to a subquadrant in the SFC mappings. Without BitShuffle, the block-entropies for both datasets are lower with SFC reorderings, though negligible for the SKA dataset ($\approx$0.05 lower). The SFCs reduce the block-entropy for the SRTM dataset without BitShuffle, compared to the Raster Scan, by approximately 0.15. However, all three SFCs have equivalent block-entropies as they have the same subquadrant extents (all are recursive with $B = 2^d$). Chunking the datasets into blocks with the same extents as the SFC subquadrants (assuming a block-entropy blocksize larger than the chunk size) would result in the same block-entropy results regardless of the SFC used. However, the block-entropies are more useful in understanding the impact of BitShuffle. By showing where the original datasets are the most redundant.

As the block-entropy for the SRTM dataset with BitShuffle is zero for the last 30 % of each file, we can conclude that the most significant 4 bits of the *int16* values are mostly redundant ($\lfloor 30\,\% \times 16\,\text{bit} \rfloor = 4\,\text{bit}$). With SFC reorderings, the median block-entropy is effectively zero for the last half of each file. This implies that at least half of the SRTM files only require one byte instead of two to represent most of their values when using an SFC and BitShuffle. This would require some metadata to store the actual value of the upper bits for specified blocks of values. With the Raster Scan, the median block-entropy reaches zero approximately one-bit later and thus would require 9 bits when BitShuffle is used.

A similar claim cannot be made for the SKA dataset as the block-entropies never drop to zero. However, the bits in the floating-point values have distinctly different entropies based on their location in the binary representation. Each IEEE-754 single-precision value in the SKA data consists of 32 bits split into a mantissa (21 bits), exponent (8 bits), and sign (1 bits). The first 71.9 % of the BitShuffled SKA files contain the mantissa, with the least-significant bits first. Even though the block-entropy begins to drop at about 60 % through the SKA files, this only accounts for the three most significant bits of the mantissa. The exponent bits are far more redundant, achieving a median block-entropy below 0.5. However, even though SFCs had little effect without BitShuffle, and on the mantissa bits with BitShuffle, the block-entropy for the exponent bits is 0.2 lower with SFC reorderings than with the Raster Scan. As the block-entropy for the mantissa bits is the maximum possible (1.0), it can be assumed that the

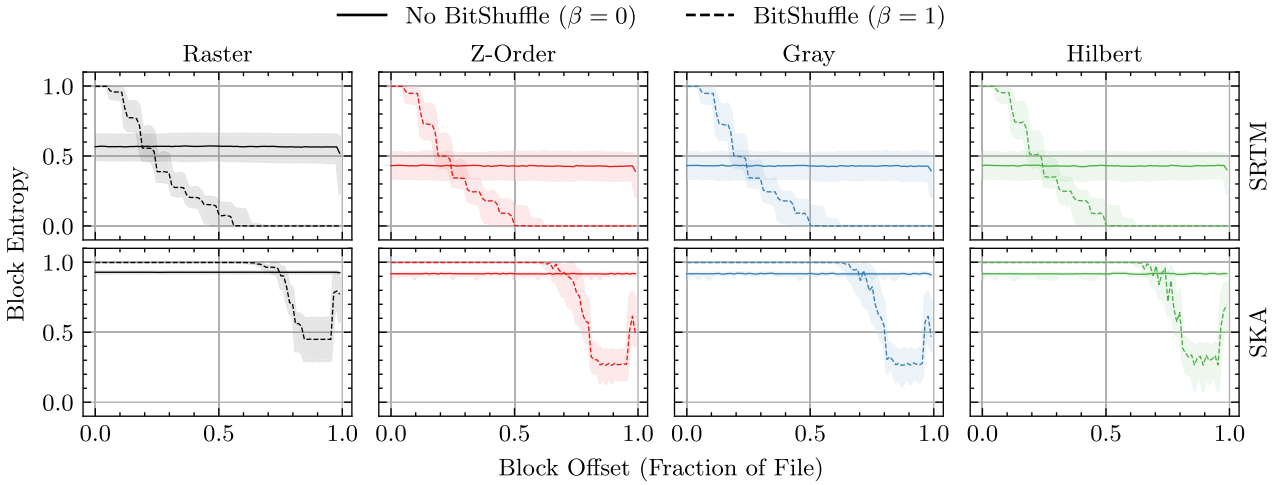

FIGURE 6. Median block-entropy for both datasets, all SFCs, and with and without BitShuffle. The top and bottom rows contain results for the SRTM dataset and SKA dataset respectively. Shaded-areas denote the inter-quartile ranges. The block-size used is 16 KiB. The entropy is calculated using byte-sized symbols and normalized to between 0 and 1.

majority of compression gains on the SKA data are concentrated in the exponent and sign bits.

Not all compression schemes would be able to exploit the lower entropies present in the preprocessed files as they use other non-entropy based algorithms. However, the file-entropies and block-entropies shown are indicative of the restructuring carried out by the SFCs and BitShuffle. Secondly, the significant reduction in block-entropy, for the SRTM dataset without BitShuffle, when SFCs are applied shows that they effectively exploit the high local-correlations of Gaussian Random Fields.

B. COMPRESSION RATIOS

For nearly all compression schemes and SFCs, the use of BitShuffle always improves compression ratios. The only two instances where this is not the case is for BZIP2 with the SRTM dataset and Huffman Encoding with the SKA dataset and the Raster Scan. Compression Ratios are more than 30 % smaller for BZIP2 and the SRTM dataset when BitShuffle is applied. Even though BitShuffle improves compression ratios for the SKA dataset with Huffman Encoding and SFC reordering, the compression scheme always results in larger dataset sizes for the SKA data. LZW and RLE also fail at compressing the SKA dataset to above-unity compression ratios. RLE only achieves $\mathcal{R}_c$ values above two for the SRTM dataset if BitShuffle is used, between 1.05 and 1.13 if not. Without BitShuffle, RLE has the worst compression ratios for the SRTM dataset whereas it is the worst for SKA dataset with and without BitShuffle. For $\beta = 0$ and all SFCs, RLE has a compression ratio of 0.8 for the SKA dataset owing to the low local correlation of the data and the specific byte encoding used by the RLE implementation. As the SKA dataset uses IEEE-754 single-precision floats, each RLE encoded symbol requires five bytes: a run length and the run value. If there are no runs with lengths greater than one, we would expect a compression ratio of $\mathcal{R}_c = 0.8$, the $\mathcal{R}_c$ achieved for the SKA dataset with RLE and no BitShuffle. This shows that RLE, with this naive encoding scheme, is wholly unsuitable for compressing the SKA dataset. Even with BitShuffle, RLE ratios are only increased by up to approximately 10 % yet still sub-unity ($\mathcal{R}_c < 1$).

Compression Ratios for the SKA dataset never exceed $\mathcal{R}_c = 1.3$. HFF, LZW, and RLE all expand the dataset size — in all cases — rather than reducing it. However, without BitShuffle, the largest compression ratio for the SKA dataset is 1.12 when using GZIP with either the Z-Order or Gray-Code Curves. For the SRTM dataset, all compression schemes achieve $\mathcal{R}_c > 1$, under all configurations. When it comes to the effect of SFC reordering versus the Raster Scan, the curves nearly always improve compression for both datasets when BitShuffle is also applied. Without BitShuffle, the only schemes improved through the use of SFC reorderings are GZIP, LZO, and LZW for the SRTM dataset and BZIP2, GZIP, and LZW for the SKA dataset. Interestingly, the Hilbert Curve gives the biggest improvement of all the SFCs for the SRTM data, when SFCs do improve $\mathcal{R}_c$ values. The Z-Order and Gray-Code Curves do decrease compression ratios by more than 5 %, without BitShuffle, in some cases: BZIP2, LZ4, LZ77, and RLE for the SRTM dataset. The only situation where the Hilbert Curve reduces $\mathcal{R}_c$ more than 5 % is with BZIP2, no BitShuffle, and the SRTM dataset.

With BitShuffle, compression ratios are increased through the use of SFC reordering anywhere between 3 % and 15.5 % for the SRTM dataset and 0.79 % and 7.56 % for the SKA dataset. These ranges are expanded further when comparing SFC reordering with BitShuffle to the Raster Scan without BitShuffle. The largest increase in compression ratio over the default unpreprocessed row-major ordering is with RLE, BitShuffle, and the Hilbert Curve on the SRTM dataset; increasing $\mathcal{R}_c$ from 1.13 to 2.46, an increase of 117.7 %. For the SKA dataset, the largest percentage is 20.39 % for both the Z-Order and Gray-Code Curves with BitShuffle and

LZ77. Without BitShuffle, the biggest percentage increase using an SFC reordering is 37.24 % and 2.19 % with the Hilbert Curve and LZW for the SRTM and SKA datasets respectively. Interestingly, the Hilbert Curve loses this lead to the other two SFCs when BitShuffle is applied to the SKA dataset. In general, the best SFCs for increasing $\mathcal{R}_c$ are the Hilbert Curve for the SRTM dataset and the Z-Order Curve for the SKA dataset. For the SKA dataset with BitShuffle, the Z-Order Curve improves $\mathcal{R}_c$ by an average of 4.73 % compared to the Raster Scan. For the SRTM dataset, where the Hilbert Curve does improve the compression ratio, $\mathcal{R}_c$ is increased on average by 13.13 % through its use.

The Z-Order and Gray-Code Curves consistently achieve the highest compression ratios for the SKA dataset with BitShuffle, exceeding $\mathcal{R}_c = 1.2$ for BZIP2, GZIP, LZ4, LZ77, and LZO. The Hilbert Curve achieves between 1.19 and 1.21 ratios for the same compression schemes. The largest compression ratio for the SKA dataset with BitShuffle, of 1.25, is achieved with GZIP and the Z-Order Curve. However, the largest absolute increase in $\mathcal{R}_c$ when applying SFC reorderings, for the SKA dataset, is with LZ77 and the Z-Order and Gray-Code Curves, not GZIP; increasing from $\mathcal{R}_c = 1.03$ ($\beta = 0$) and $\mathcal{R}_c = 1.17$ ($\beta = 1$) to 1.24 ($\beta = 1$). The largest absolute increase in $\mathcal{R}_c$ for the SRTM dataset is also for LZ77, but instead with the Hilbert Curve, increasing from 2.96 ($\beta = 0$) and 3.7 ($\beta = 1$) to 4.03 ($\beta = 1$). Without BitShuffle, compression ratios are increased by less than 0.2 for the SKA dataset. However, the largest equivalent increase for the SRTM dataset is with LZW, from $\mathcal{R}_c = 2.39$ to $\mathcal{R}_c = 3.28$.

For both datasets, the Lempel-Ziv based compression schemes perform similarly. Huffman Encoding and RLE perform the worst while BZIP2 performs the best for the SRTM dataset and on par with the Lempel-Ziv schemes for the SKA dataset. The Burrows-Wheeler Transform (BWT) exploits strong local correlation between adjacent values by implementing a reversible pseudo-sort. If the local correlation is weak, the BWT would be less effective. This is most likely why BZIP2, in which BWT is the *core* component, performs worse for the SKA data.

The aforementioned literature for DEM data achieves higher compression ratios than those shown here. Scarmana and McDougall achieve compression ratios similar to those in the work presented here with LZW and GZIP/DEFLATE but their Hilbert Curve based compression scheme using DPCM achieves compression ratios above $\mathcal{R}_c = 7$, higher than BZIP2 for the SRTM dataset [42]. None of the compression schemes investigated in this paper use predictive coding, the primary technique in DPCM. Furthermore, the dataset used by Scarmana and McDougall is not the same as the SRTM dataset, indicating that the two results are not entirely comparable. The average compression ratio found by Rane and Sapiro for the JPEG-LS compression scheme on DEM data is $\mathcal{R}_c = 11.75$ [45]. This is significantly higher than compression ratios achieved here and in the work by Scarmana and McDougall [42]. Rane and Sapiro investigate only JPEG-LS, the lossless and near-lossless JPEG encoding scheme [45]. Scarmana and McDougall also investigate JPEG-LS with their dataset, but only achieve an average compression ratio of $\mathcal{R}_c = 5.97$ [42], a value much lower than that achieved by Rane and Sapiro for 16 bit integer DEM data [45]. This raises the question of how these approaches to compressing DEM data would perform on a common dataset.

For the SKA dataset, the lossy BitShuffle technique defined by Masui *et al.* attains compression ratios of approximately $\mathcal{R}_c 3.58$, higher than that attained through lossless compression of the SKA dataset [46]. Though, this significant performance is achieved with lossy compression, which is outside the scope of this work. But it does highlight that radio astronomy data is far more redundant when some precision is lost for the sake of size requirements. The work by Zheng *et al.*, in compressing astronomy data using an *effective bit-width* transform with GZIP, shows that there is still room for lossless compression schemes for similar datasets [49]. Even so, their dataset, and that used by Masui *et al.*, contains integer data and not floating-point data as is the case with the SKA dataset. Regardless of the comparative performance between previous works and the results conveyed here, it is evident that astronomy data requires some bit-level manipulation to compress effectively. Whether this is BitShuffle, an *effective bit-width*, or an alternative scheme, compression is dependent on the underlying bit-level statistics of the data type and dataset.

C. COMPRESSION AND DECOMPRESSION SPEEDS

The compression and decompression speeds observed are somewhat correlated with the compression ratios achieved. This relationship is evident in Fig. 7, where lower $\mathcal{R}_c$ values have higher compression and decompression rates. The numerical results in Table 4 also show how different configurations change processing speeds. Decompression is generally slower than compression by about 20 %, for all compression schemes except RLE, while decompression is up to 26 % and 42 % slower for the SRTM and SKA datasets respectively. However, RLE is the fastest compression scheme in all cases, as no dictionary or encoding table is needed to process the data. Compression speeds are predominantly slower for the SRTM dataset when some preprocessing is applied (SFCs and/or BitShuffle). The application of SFC reordering to the SRTM dataset only improves compression and decompression speeds for GZIP and LZO with BitShuffle. When compared to the unpreprocessed configuration, their improvements are significantly diminished, if not negated entirely. Without BitShuffle, the Raster Scan always performs faster than the other three SFCs for the SRTM dataset.

Nearly all compression schemes, applied to the SKA dataset, are slowed down through the use of BitShuffle. The impact is different for the SRTM dataset where it sometimes benefits the overall speed. In many cases, it has very little

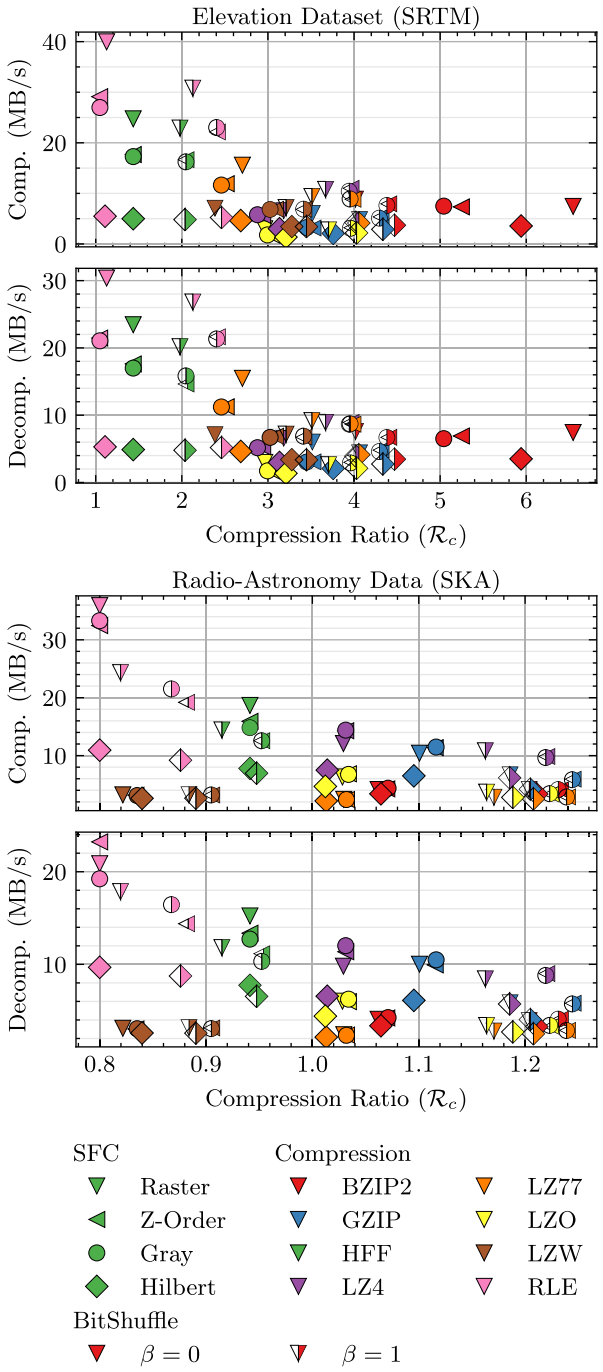

FIGURE 7. Compression and decompression speed versus compression ratio for all compression schemes, SFCs, and with and without BitShuffle. Note that the application of BitShuffle is denoted by $\beta = 1$.

impact on compression and decompression speeds for the SRTM dataset. It is possible that the reduced shuffle rate for three-dimensional floating-point data shown in Fig. 5 is the cause for the lower speeds in the SKA dataset when BitShuffle is applied. However, the time elapsed to conduct BitShuffle is a few milliseconds at most and is independent of the compression scheme used. Therefore, any difference in compression and decompression speed between compression techniques, for a given dataset and β, must be because of the compression scheme and its interaction with BitShuffle and the SFCs, and not the BitShuffle implementation itself.

Though SFC reorderings do not positively impact compression speeds without BitShuffle for the SRTM dataset, the SKA dataset does benefit from their usage. When applying the Gray-Code Curve without BitShuffle to the SKA dataset, the compression and decompression speeds of LZ4 are increased by 19 % and 22.5 % respectively. This benefit is diminished when BitShuffle is applied, improving decompression speeds by under 10 % but slowing compression down by 10.6 %. However, the introduction of BitShuffle to any configuration for the SKA dataset nearly always reduces speeds. The only exceptions are LZ77 and LZW which are approximately 0.4 MB/s and 0.04 MB/s faster respectively. For all compression schemes and configurations, the Hilbert Curve is always the slowest. This can be partially attributed to the significantly slower mapping rate shown in Fig. 4. The total runtime for each compression scheme has a roughly-constant delay when using the Hilbert Curve compared to the Raster Scan and other two SFCs. For the SKA and SRTM datasets, this offset is 400 s to 500 s and approximately 1000 s respectively. The shorter delay when using the Hilbert Curve for the SKA dataset is caused by the approximately 1.5×10^5 Hz higher mapping rate for the three-dimensional Hilbert mapping with a sidelength of 2^7 compared to the two-dimensional Hilbert Mapping with a sidelength of 2^{11}.

For both datasets, the second fastest compression scheme is Huffman Encoding with the Raster Scan. This may be owing to the generation and processing of the Huffman tree but further research is required to identify the cause. Huffman Encoding is slowed down through the application of BitShuffle for both datasets, even with the Raster Scan. LZ77 is the third fastest scheme for the SRTM dataset but the slowest for the SKA dataset. This may be owing to the lack of an optimized dictionary search algorithm in the LZ77 implementation used, as the other LZ77-based schemes (GZIP, LZ4, and LZO) are faster. Of the compression schemes that achieve greater than unity compression ratios for the SKA dataset, the fastest is LZ4 with BitShuffle and the Z-Order Curve. In fact, the Z-Order and Gray-Code Curves improve the compression and decompression speeds of BZIP2 and LZ4 for the SKA dataset, when compared to the Raster Scan.

When compared to the unpreprocessed configurations, the biggest percentage increase in speeds for the SRTM and SKA datasets are for LZ4 with the Z-Order Curve ($\beta = 1$) and the Gray-Code Curve ($\beta = 0$) respectively. This is also the case for the biggest absolute increase in compression and decompression speeds, ignoring RLE for the SKA dataset as it does not achieve a reduction in dataset size. With LZ4, BitShuffle, and the Z-Order Curve, the SRTM dataset is compressed at 11 MB/s and decompressed at 8.92 MB/s. With LZ4 and the Gray-Code Curve, the SKA dataset is compressed at 14.4 MB/s and decompressed at 12 MB/s.

However, LZ4 only achieves a compression ratio of 1.03 for the SKA dataset without BitShuffle. With BitShuffle, the Z-Order Curve is better, with compression and decompression speeds of 9.84 MB/s and 9.01 MB/s respectively. This is slower than if the Raster Scan was used without BitShuffle. If BitShuffle is to be used on the SRTM dataset, the benefit of SFC reordering on the speed of LZ4 is lost, indicating that the majority of the speed improvement comes from the usage of BitShuffle and not SFCs. However, the speeds of GZIP and LZO ($\beta = 1$) are increased anywhere between 5 % and 15 % by applying the Z-Order or Gray-Code Curve, with the former introducing the biggest improvement.

The speed of LZ77 on the SKA dataset is increased by 15 % to 20 % by using BitShuffle, with no substantial improvement through the use of SFC reorderings. The accompanying speeds never reach 3 MB/s. Without BitShuffle, LZ77 is slower, more so when SFCs are applied. However, when LZ77 is used on the SRTM dataset, BitShuffle has the opposite effect, slowing down speeds with the Raster Scan always achieving the fastest compression and decompression. The significant improvement to compression ratios for LZW by using SFC reordering on the SRTM dataset does not translate to increases in speed. Compression and decompression speeds are 2 % to 6 % slower without BitShuffle and 4 % to 7 % slower with BitShuffle than when using the Raster Scan.

Within the same literature covered at the end of Section IV-B, only three explore compression speeds. The Hilbert Curve and DPCM-based compression scheme by Scarmana and McDougall has a compression speed of approximately 4 MB/s [42], which is around the middle for all compression schemes in Fig. 7 for the SRTM dataset. Their compression speeds for LZW and DEFLATE are comparable to those achieved in this work, though GZIP (DEFLATE) is slightly slower for the SRTM dataset. This may be because the results for GZIP in Table 4 and Fig. 7 use the highest compression level and thus take longer than with the default DEFLATE parameters from PNG, used by Scarmana and McDougall.

Comparing compression speeds for the SKA dataset, with the results obtained by Zheng *et al.* [49] and Masui *et al.* [46], is not appropriate as they use hardware configurations that differ greatly from that used here. Masui *et al.* execute their tests on data files stored entirely in memory, removing any I/O overhead of a permanent storage medium such as an SSD [46]. This is why they are able to achieve a write-speed of 749 MiB/s. Zheng *et al.* do not store their data files in memory, but they do use an FPGA as an external accelerator [49]. They do experiment with executing their technique on a CPU alone, but it is nearly seven times slower than BitShuffle with LZ4.

V. DISCUSSION

SFC reordering has a significant impact on file-entropy and block-entropy for the SRTM dataset, without BitShuffle. Block-entropy values are on average 0.1 lower with SFCs than with the Raster Scan. When BitShuffle is applied, this impact is significantly reduced as BitShuffle is the primary reason for the reduction in entropy. However, the block-entropy for the most significant bits is effectively zero for at least half of the files when SFC orderings are used. The block-entropy with BitShuffle is about 0.15 lower with SFCs than with the Raster Scan. The high redundancy in the most significant bits of the SRTM data is caused by its high local-correlation and the integer representation used. The opposite effect is observed in the SKA dataset, with SFCs having a greater effect with BitShuffle than without. SFC reordering has little to no effect on block-entropies for the SKA dataset when BitShuffle is not applied, with block-entropies just above 0.9. One reason for this is the low local-correlation in the SKA data, indicating that values are more dissimilar than alike within a small neighbouring region. The floating-point values are compact in their structure, but there are redundant sections within the binary representation. BitShuffle allows the components of the single-precision floating-point values with the smallest variation to be compressed: the exponent and sign bits. With BitShuffle and the Raster Scan, the block-entropy for the exponent and sign bit regions of the SKA dataset is around 0.4 lower than without BitShuffle. SFC reordering lowers this by a further 0.2, a massive reduction over the block-entropy without preprocessing. However, the mantissa makes up 72 % of each BitShuffled SKA file, resulting in file-entropies above 0.85 with SFC reordering and above 0.9 with the Raster Scan.

Of the three SFCs, the Hilbert Curve achieves the best compression ratios for the SRTM dataset without BitShuffle. However, the Raster Scan has the largest $\mathcal{R}_c$ for all compression schemes except for GZIP, LZO, and LZW. With BitShuffle applied to the SRTM dataset, the Raster Scan always has the smallest compression ratio while the Hilbert Curve has the largest; though it misses out by 0.01 to the Z-Order and Gray-Code Curves for Huffman Encoding. For the SKA dataset, the Hilbert Curve only achieves the largest compression ratio with LZW and no BitShuffle but lands up increasing the dataset size anyway ($\mathcal{R}_c = 0.84$). Regardless of whether BitShuffle is used, the Z-Order and Gray-Code Curves consistently achieve compression ratios larger than if the Raster Scan was used. Without BitShuffle, the largest $\mathcal{R}_c$ value is 1.12 with GZIP and either of the Z-Order or Gray-Code Curves. None of the other compression schemes achieve more than $\mathcal{R}_c = 1.1$ without BitShuffle. The top compressors for the SKA dataset — with BitShuffle — are BZIP2, GZIP, LZ4, LZ77, and LZO (all within compression ratios of $1.22 \leq \mathcal{R}_c \leq 1.25$). Without BitShuffle, the same compressors achieve maximum compression ratios of 1.07, 1.12, 1.03, 1.03, 1.03 respectively. The top compressors for the SRTM dataset are BZIP2 and GZIP with BitShuffle ($\mathcal{R}_c = 4.47$ and $\mathcal{R}_c = 4.33$) and without BitShuffle ($\mathcal{R}_c = 6.54$ and $\mathcal{R}_c = 3.76$).

Even though SFC reorderings have a generally positive impact on compression ratios, in only a few cases are the compression or decompression speeds also increased.

The time requirements of the Hilbert Curve algorithm and implementation used [77] results in impractical speed reductions, for both compression and decompression, across the board. Given that DEM data are not found in file-formats that support BitShuffle, the best compression configuration does appear to be BZIP2 with the Raster Scan and without BitShuffle: achieving a compression ratio of 6.54 and compression and decompression speeds of 7.43 MB/s. Of the compression schemes investigated, only Huffman Encoding, LZ77, LZW, and RLE are in the DEM file-formats identified in Table 2. Of these four schemes, LZW is improved the most by SFC reordering (Z-Order Curve), increasing the compression ratio from 2.39 to 3.09 while effectively maintaining compression and decompression speeds within a margin of 0.24 MB/s. If a faster Hilbert Curve mapping implementation were to be integrated, a compression ratio of 3.28 could be obtained without the substantial speed penalty present in these results. Interestingly, GZIP is improved in all regards through the use of the Gray-Code Curve when applied to the SKA dataset without BitShuffle. The fastest scheme for the SRTM dataset is RLE followed by Huffman Encoding. In fact, RLE achieves faster compression and larger compression ratios than Huffman Encoding, when BitShuffle is used. The general lacklustre speeds with preprocessing are more than likely because of implementation specific details as is the case with the Hilbert Curve and BitShuffle on three-dimensional floating-point data. Further research in this regard is needed to determine how the speeds can be brought to be on par with the Raster Scan while maintaining the better compression ratios measured. If the Hilbert Curve and BitShuffle implementations are optimized for faster computation, then the compression ratio advantages they provide can be exploited without the speed penalty found in this work.

The SKA dataset is highly incompressible, as shown by the low compression ratios in Table 4. As is evident in the literature, BitShuffle has a significant impact on compression ratios, achieving a maximum of $\mathcal{R}_c = 1.25$ with GZIP and the Z-Order Curve compared to the maximum without BitShuffle of $\mathcal{R}_c = 1.12$ with the same scheme and SFC. Unlike for the SRTM data, the Hilbert Curve is never the best compressor for the SKA Dataset, with or without BitShuffle. Secondly, the Raster Scan is never the best compressor by itself, always tied with the Z-Order and Gray-Code Curves. Furthermore, the Raster Scan is always the worst ordering for compression with BitShuffle for both datasets. This shows that the *best* SFC reordering scheme is different for these two types of datasets. The strong differences in their autocorrelations, coupled with their data types, is the clearest reason behind why the Hilbert Curve is best for the SRTM data and the Z-Order and Gray-Code Curves are best for the SKA data. As is shown by Mokbel, Aref, and Kamel, the Hilbert Curve has no bias to any given dimension whereas the Raster Scan, Z-Order, and Gray-Code curves have strong biases towards specific dimensions [56]. Given the SKA data's strong correlation in the time-dimension, compared to the channel and correlator dimensions, it is natural to ask whether a variant of the Raster Scan that traverses the dimensions in a different order would result in *better* compression performance than the orderings investigated here.

VI. FUTURE WORK

As discussed in Sections IV and V, the implementation of the Hilbert Curve mapping is the biggest opportunity for improvement as its usage improves compression ratios for the SRTM dataset but is significantly slower owing to the slow mapping rate. The mapping implementation for the Z-Order and Gray-Code Curves exploits AVX2 and BMI2 x86 instruction sets [76] whereas the Hilbert Curve implementation does not [78]. As is previously highlighted, the Z-Order and Gray-Code Curves perform best for the SKA dataset. However, given the stronger autocorrelation in the time-dimension, there is the possibility that a variant of the Raster Scan or another SFC would result in better compression performance if it can exploit this dimensional bias. Mokbel, Aref, and Kamel develop an appropriate metric to quantify any bias in d-dimensional orderings, called a description vector [56]. If further work is conducted on reordering radio-astronomy data for the purpose of compression, the description vector is an appropriate and necessary tool. The compression ratios achieved are good, but comparing these results to those in the literature indicates that there may be other compression schemes that are better suited for the datasets with SFC reorderings [42], [45], [46], [49]. Applying the following methodology to other compression schemes would expand upon the knowledge of how such reorderings effect compression performance. The most interesting candidates, in the authors' opinions, are arithmetic encoding, predictive coding (such as DPCM), and domain specific lossy compression schemes. Furthermore, compressing floating-point DEM data using this methodology would help identify the impact of different data types on similar IGRF data-models.

VII. CONCLUSION

The application of SFC reordering to DEM and radio-astronomy data gives varying results that are dependent more on the compression scheme than the curve used and whether BitShuffle is applied. SFCs are shown to reduce file-entropy and block-entropy for the SRTM dataset (without BitShuffle) and the SKA dataset (with BitShuffle). However, there is little difference between the Z-Order, Gray-Code, and Hilbert Curves, as they are all Recursive Space-Filling Curves with the same subquadrants and thus data extents. BitShuffle is shown to improve compression ratios across the board with varying speeds. The Z-Order and Gray-Code Curves improve compression ratios and maintain compression speeds for both datasets, when compared to the Raster Scan. However, the Hilbert Curve gives larger compression ratios for the SRTM dataset. Using the three curves, the compression ratio of the SRTM dataset using LZW is increased from 2.39 to 3.09 while maintaining a speed of around 7 MB/s using the Z-Order Curve. GZIP and LZ4, combined with BitShuffle and the Z-Order Curve, achieve compression ratios of

1.24 and 1.22 for the SKA Dataset, with average speeds of 5.87 MB/s and 9.43 MB/s respectively. In fact, the compression performance of GZIP on the SKA dataset, without BitShuffle, is improved in all regards by using the Gray-Code Curve ($\mathcal{R}_c$ from 1.1 to 1.2 and speeds from approximately 10 MB/s to approximately 11 MB/s).

The Hilbert Curve consistently achieves higher compression ratios for the SRTM data with and without BitShuffle but loses to the Z-Order and Gray-Code Curves for the SKA dataset. However, compression speeds are significantly biased towards the standard Row-Major Order for the SRTM dataset, with the fastest configuration on the SKA dataset varying between Row-Major Order, the Z-Order Curve, and the Gray-Code Curve. Furthermore, the Hilbert Curve mapping implementation introduces a significant runtime penalty — compared to the Row-Major, Z-Order, and Gray-Code Curves — making its usage in these contexts impractical unless a faster mapping algorithm is identified and integrated. The Z-Order and Gray-Code Curves show consistent improvement to compression ratios and varying improvement to processing speeds over the Raster Scan, for the SKA dataset with BitShuffle, indicating that alternative orderings do benefit compression performance for radio-astronomy data. If a faster Hilbert Mapping algorithm is identified, its use as a reordering scheme for DEM data shows significant promise for improving already existing compression schemes. Some additional speed improvements may be gained by further optimizing the BitShuffle implementation similarly to the Hilbert Curve implementation.

ACKNOWLEDGMENT

The authors would like to thank the South African Radio-Astronomy Observatory for providing and allowing the use of the SKA dataset as well as the United-States Geological Survey for freely providing the Shuttle-Radio Topographic Mission data through their EarthExplorer tool.

REFERENCES

[1] H. V. Jagadish, "Linear clustering of objects with multiple attributes," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*. New York, NY, USA: ACM, 1990, pp. 332–342, doi: 10.1145/93597.98742.

[2] J. K. Lawder and P. J. H. King, "Using space-filling curves for multi-dimensional indexing," in *Proc. Adv. Databases 17th Brit. Nat. Conf. Databases (BNCOD)*, B. Lings and K. Jeffery, Eds. Berlin, Germany: Springer, Jul. 2000, pp. 20–35, doi: 10.1007/3-540-45033-5_3.

[3] OSGeo Project. (Feb. 2020). *PostGIS 3.0.1 Dev Manual*. [Online]. Available: https://postgis.net/stuff/postgis-3.0.pdf

[4] J. Qi, G. Liu, C. S. Jensen, and L. Kulik, "Effectively learning spatial indices," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 2341–2354, Aug. 2020, doi: 10.14778/3407790.3407829.

[5] X. Guan, P. van Oosterom, and B. Cheng, "A parallel N-dimensional space-filling curve library and its application in massive point cloud management," *ISPRS Int. J. Geo-Inf.*, vol. 7, no. 8, p. 327, Aug. 2018. [Online]. Available: https://www.mdpi.com/2220-9964/7/8/327

[6] M. Pavlovic, K.-N. Bastian, H. Gildhoff, and A. Ailamaki, "Dictionary compression in point cloud data management," *ACM Trans. Spatial Algorithms Syst.*, vol. 5, no. 1, pp. 1–25, Jun. 2019, doi: 10.1145/3299770.

[7] D. Holzmüller, "Efficient neighbor-finding on space-filling curves," Ph.D. dissertation, Univ. Stuttgart, Stuttgart, Germany, Oct. 2017. [Online]. Available: http://arxiv.org/abs/1710.06384

[8] H. Haverkort and F. van Walderveen, "Locality and bounding-box quality of two-dimensional space-filling curves," *Comput. Geometry*, vol. 43, no. 2, pp. 131–147, Feb. 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925772109000765

[9] H. Lian, W. Qiu, D. Yan, Z. Huang, and P. Tang, "Efficient and secure K-nearest neighbor query on outsourced data," *Peer-to-Peer Netw. Appl.*, vol. 13, no. 6, pp. 2324–2333, Nov. 2020, doi: 10.1007/S12083-020-00909-2.

[10] S. Sieranoja and P. Fränti, "Constructing a high-dimensional kNN-graph using a Z-order curve," *ACM J. Express Algorithmics*, vol. 23, pp. 1.9:1–1.9:21, Oct. 2018, doi: 10.1145/3274656.

[11] U. Sakoglu, L. Bhupati, N. Beheshti, N. Tsekos, and L. Johnsson, "An adaptive space-filling curve trajectory for ordering 3D datasets to 1D: Application to brain magnetic resonance imaging data for classification," in *Computational Science—ICCS* (Lecture Notes in Computer Science), V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, and J. Teixeira, Eds. Cham, Switzerland: Springer, 2020, pp. 635–646.

[12] P. Tsinganos, B. Cornelis, J. Cornelis, B. Jansen, and A. Skodras, "A Hilbert curve based representation of sEMG signals for gesture recognition," in *Proc. Int. Conf. Syst., Signals Image Process. (IWSSIP)*, Jun. 2019, pp. 201–206.

[13] P. Tsinganos, B. Cornelis, J. Cornelis, B. Jansen, and A. Skodras, "Hilbert sEMG data scanning for hand gesture recognition based on deep learning," *Neural Comput. Appl.*, vol. 33, no. 7, pp. 2645–2666, Apr. 2021, doi: 10.1007/s00521-020-05128-7.

[14] Z. Ren, G. Chen, and W. Lu, "Space filling curve mapping for malware detection and classification," in *Proc. 3rd Int. Conf. Comput. Sci. Softw. Eng.* New York, NY, USA: Association for Computing Machinery, May 2020, pp. 176–180, doi: 10.1145/3403746.3403924.

[15] C. Bohm, M. Perdacher, and C. Plant, "A novel Hilbert curve for cache-locality preserving loops," *IEEE Trans. Big Data*, early access, May 4, 2018, doi: 10.1109/TBDATA.2018.2830378.

[16] T. Bially, "Space-filling curves: Their generation and their application to bandwidth reduction," *IEEE Trans. Inf. Theory*, vol. IT-15, no. 6, pp. 658–664, Nov. 1969.

[17] H. K. Dai and H. C. Su, "Clustering performance of 3-dimensional Hilbert curves," in *Algorithmic Aspects in Information and Management* (Lecture Notes in Computer Science), Q. Gu, P. Hell, and B. Yang, Eds. Cham: Springer, 2014, pp. 299–311.

[18] N. J. Rose. (Aug. 2015). *Hilbert-Type Space-Filling Curves*. [Online]. Available: http://www4.ncsu.edu/~njrose/pdfFiles/HilbertCurve.pdf

[19] G. Peano, "Sur une courbe, qui remplit toute une aire plane," *Mathematische Annalen*, vol. 36, no. 1, pp. 157–160, Mar. 1890. [Online]. Available: https://link.springer.com/article/10.1007/BF01199438

[20] H. L. Lebesgue, "Lecons sur l'integration et la recherche des fonctions primitives, professees au college de France (deuxiemeedition)," in *Collection de Monographies Sur la Theorie des Fonctions*. Paris, France: Gauthier-Villars et cie, 1928.

[21] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," Int. Bus. Mach., Ottawa, ON, Canada, Tech. Rep., 1966.

[22] D. Hilbert, "Ueber die stetige Abbildung einer line auf ein Flächenstück," *Math. Ann.*, vol. 38, no. 3, pp. 459–460, Sep. 1891. [Online]. Available: https://link.springer.com/article/10.1007/BF01199431

[23] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer, "Space-filling curves and their use in the design of geometric data structures," *Theor. Comput. Sci.*, vol. 181, no. 1, pp. 3–15, Jul. 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397596002599

[24] C. Faloutsos and S. Roseman, "Fractals for secondary key retrieval," in *Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. Princ. Database Syst. (PODS)*. Philadelphia, PA, USA: ACM Press, 1989, pp. 247–252. [Online]. Available: http://portal.acm.org/citation.cfm?doid=73721.73746

[25] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the Hilbert space-filling curve," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 1, pp. 124–141, Jan. 2001.

[26] A. Lempel and J. Ziv, "Compression of two-dimensional data," *IEEE Trans. Inf. Theory*, vol. IT-32, no. 1, pp. 2–8, Jan. 1986.

[27] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. IT-23, no. 3, pp. 337–343, May 1977.

[28] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 5, pp. 530–536, Sep. 1978.

[29] F. Pinciroli, C. Combi, G. Pozzi, M. Negretto, L. Portoni, and G. Invernizzi, "A Peano-Hilbert derived algorithm for compression of angiocardiographic images," in *Proc. Comput. Cardiol.*, Sep. 1991, pp. 81–84.

[30] F. J. Merkl, "Binary image compression using run length encoding and multiple scanning techniques," M.S. thesis, Rochester Inst. Technol., Rochester, NY, USA, 1988. [Online]. Available: https://scholarworks.rit.edu/theses/184

[31] Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, Jun. 1984.
[32] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.
[33] J.-Y. Liang, C.-S. Chen, C.-H. Huang, and L. Liu, "Lossless compression of medical images using Hilbert space-filling curves," *Computerized Med. Imag. Graph.*, vol. 32, no. 3, pp. 174–182, Apr. 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S089561110700167X
[34] A. Abdollahi, N. Bruce, S. Kamali, and R. Karim, "Lossless image compression using list update algorithms," in *String Processing and Information Retrieval* (Lecture Notes in Computer Science), N. R. Brisaboa and S. J. Puglisi, Eds. Cham, Switzerland: Springer, 2019, pp. 16–34.
[35] J. A. Provine and R. M. Rangayyan, "Lossless compression of peanoscanned images," *J. Electron. Imag.*, vol. 3, no. 2, pp. 176–181, Apr. 1994. [Online]. Available: https://www.spiedigitallibrary.org/journals/journal-of-electronic-imaging/volume-3/issue-2/0000/Lossless-compression-of-Peanoscanned-images/10.1117/12.171928.short
[36] T. Ouni, A. Lassoued, and M. Abid, "Gradient-based space filling curves: Application to lossless image compression," in *Proc. IEEE Int. Conf. Comput. Appl. Ind. Electron. (ICCAIE)*, Dec. 2011, pp. 437–442.
[37] T. Ouni, A. Lassoued, and M. Abid, "Lossless image compression using gradient based space filling curves (G-SFC)," *Signal, Image Video Process.*, vol. 9, no. 2, pp. 277–293, Feb. 2015, doi: 10.1007/s11760-013-0435-4.
[38] CompuServe Incorporated. (Jul. 1990). *Graphics Interchange Format (89a): Programming Reference*. [Online]. Available: https://www.w3.org/Graphics/GIF/spec-gif89a.txt
[39] T. Boutell. (1997). *PNG (Portable Network Graphics) Specification Version 1.0*. [Online]. Available: https://www.rfc-editor.org/info/rfc2083
[40] P. Deutsch, "DEFLATE compressed data format specification version 1.3," RFC Editor, Internet Soc., VA, USA, Tech. Rep. RFC1951, May 1996. [Online]. Available: https://www.rfc-editor.org/info/rfc1951
[41] R. Pračko and J. Polec, "DPCM application to images scanned by SFC method," *J. Electr. Eng.*, vol. 58, no. 3, p. 4, Nov. 2007.
[42] G. Scarmana and K. McDougall, "An application of space filling curves to digital terrain models," in *Proc. 16th Int. Congr. Mine Surveying, Connecting Educ. Ind.*, vol. 1. Toowoomba, QLD, Australia: Ellipsis Media, 2016, pp. 113–117.
[43] A. Quin and Y. Yanagisawa, "On data compaction of scanning curves," *Comput. J.*, vol. 32, no. 6, pp. 563–566, Dec. 1989. [Online]. Available: https://academic.oup.com/comjnl/article/32/6/563/341695
[44] N. Memon, D. L. Neuhoff, and S. Shende, "An analysis of some common scanning techniques for lossless image coding," *IEEE Trans. Image Process.*, vol. 9, no. 11, pp. 1837–1848, Nov. 2000.
[45] S. D. Rane and G. Sapiro, "Evaluation of JPEG-LS, the new lossless and controlled-lossy still image compression standard, for compression of high-resolution elevation data," *IEEE Trans. Geosci. Remote Sens.*, vol. 39, no. 10, pp. 2298–2306, Oct. 2001.
[46] K. Masui, M. Amiri, L. Connor, M. Deng, M. Fandino, C. Höfer, M. Halpern, D. Hanna, A. D. Hincks, G. Hinshaw, J. M. Parra, L. B. Newburgh, J. R. Shaw, and K. Vanderlinde, "A compression scheme for radio data in high performance computing," *Astron. Comput.*, vol. 12, pp. 181–190, Sep. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2213133715000694
[47] Y. Collet. (Jan. 2020). *LZ4*. [Online]. Available: https://github.com/lz4/lz4
[48] R. Nan, D. Li, C. Jin, Q. Wang, L. Zhu, W. Zhu, H. Zhang, Y. Yue, and L. Qian, "The five-hundred-meter aperture spherical radio telescope (FAST) project," *Int. J. Modern Phys. D*, vol. 20, no. 6, pp. 989–1024, Jun. 2011, doi: 10.1142/S0218271811019335.
[49] Y. Zheng, Y. Zhu, Y. Song, T. Nan, and W. Li, "A lossless astronomical data compression scheme with FPGA acceleration," in *Proc. 32nd IEEE Int. System-on-Chip Conf. (SOCC)*, Sep. 2019, pp. 45–49.
[50] J. R. Seward. (Jan. 2002). *BZIP2 and LibBZIP2*. [Online]. Available: https://www.sourceware.org/bzip2/
[51] P. Deutsch, "GZIP file format specification version 4.3," RFC Editor, Internet Soc., VA, USA, Tech. Rep. RFC1952, May 1996. [Online]. Available: https://www.rfc-editor.org/info/rfc1952
[52] M. F. Oberhumer. (2008). *LZO—A Real-Time Data Compression Library*. [Online]. Available: https://www.oberhumer.com/opensource/lzo/
[53] H. Sagan, *Space-Filling Curves* (Universitext Series). New York, NY, USA: Springer-Verlag, 1994. [Online]. Available: https://books.google.co.za/books?id=oEFAAQAAIAAJ
[54] M. Bader, *Space-Filling Curves: An Introduction With Applications in Scientific Computing*, (Texts in Computational Science and Engineering), no. 9. Heidelberg, Germany: Springer, 2013.
[55] C. Gotsman and M. Lindenbaum, "On the metric properties of discrete space-filling curves," in *Proc. 12th IAPR Int. Conf. Pattern Recognit. Conf. B, Comput. Vis. Image Process.*, vol. 3, Oct. 1994, pp. 98–102.
[56] M. F. Mokbel, W. G. Aref, and I. Kamel, "Analysis of multi-dimensional space-filling curves," *GeoInformatica*, vol. 7, no. 3, pp. 179–209, Sep. 2003. [Online]. Available: https://link.springer.com/article/10.1023/A:1025196714293
[57] F. Gray, "Pulse code communication," U.S. Patent 2 632 058 A, Mar. 17, 1953. [Online]. Available: https://patents.google.com/patent/US2632058A/en
[58] J. Skilling, "Programming the Hilbert curve," in *Proc. AIP Conf.*, Apr. 2004, vol. 707, no. 1, pp. 381–387, doi: 10.1063/1.1751381.
[59] T. G. Farr, P. A. Rosen, E. Caro, R. Crippen, R. Duren, S. Hensley, M. Kobrick, M. Paller, E. Rodriguez, L. Roth, D. Seal, S. Shaffer, J. Shimada, J. Umland, M. Werner, M. Oskin, D. Burbank, and D. Alsdorf, "The shuttle radar topography mission," *Rev. Geophys.*, vol. 45, no. 2, May 2007, doi: 10.1029/2005rg000183.
[60] USGS. (2018). *USGS EarthExplorer*. [Online]. Available: https://earthexplorer.usgs.gov/
[61] L. Hawker, J. Rougier, J. Neal, P. Bates, L. Archer, and D. Yamazaki, "Implications of simulating global digital elevation models for flood inundation studies," *Water Resour. Res.*, vol. 54, no. 10, pp. 7910–7928, Oct. 2018, doi: 10.1029/2018WR023279.
[62] F. J. Aguilar, M. A. Aguilar, J. L. Blanco, A. Nemmaoui, and A. M. G. Lorca, "Analysis and validation of grid DEM generation based on Gaussian Markov random field," *ISPRS-Int. Arch. Photogramm., Remote Sens. Spatial Inf. Sci.*, vol. XLI-B2, pp. 277–284, Jun. 2016. [Online]. Available: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B2/277/2016/
[63] SARAO. (Feb. 2020). *SARAO Archive*. [Online]. Available: https://archive.sarao.ac.za/
[64] P. E. Dewdney, P. J. Hall, R. T. Schilizzi, and T. J. L. W. Lazio, "The square kilometre array," *Proc. IEEE*, vol. 97, no. 8, pp. 1482–1496, Aug. 2009. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5136190/
[65] The HDF Group. (1997). *Hierarchical Data Format, Version 5*. [Online]. Available: http://www.hdfgroup.org/HDF5/
[66] I. Sihlangu. (2019). *The MeerKAT Radio Frequency Interference Environment*. [Online]. Available: https://open.uct.ac.za/handle/11427/31748
[67] *IEEE Standard for Floating-Point Arithmetic*, Standard IEEE Std 754-2019 (Revision IEEE 754-2008), Microprocessor Standards Committee, Jul. 2019, pp. 1–84.
[68] Open Geospatial Consortium. (Sep. 2019). *OGC GeoTIFF Standard*. [Online]. Available: http://www.opengis.net/doc/IS/GeoTIFF/1.1
[69] W. Büscher. (Oct. 2020). *Spectrum Lab Manual—Spectrum Displays*. [Online]. Available: https://www.qsl.net/dl4yhf/speclab/specdisp.htm#waterfall
[70] J. Tsai. (Mar. 2016). *BZIP2: Format Specification*. [Online]. Available: https://github.com/dsnet/compress/blob/master/doc/bzip2-format.pdf
[71] Canonical Ltd. (2020). *Ubuntu—Software Packages in 'Bionic' Subsection Utils*. [Online]. Available: https://packages.ubuntu.com/bionic/utils/
[72] G. Lampert. Nov. 2020. *Compression Agorithms*. [Online]. Available: https://github.com/glampert/compression-algorithms
[73] I. Tkatchev. (Jun. 2020). *Yet Another LZ77 (Yalz77)*. [Online]. Available: https://github.com/ivan-tkatchev/yalz77
[74] The HDF Group. (2006). *HDF5 Filters*. [Online]. Available: https://portal.hdfgroup.org/display/support/Filters
[75] O. Tange, "GNU parallel—The command-line power tool," *USENIX Mag.*, vol. 36, no. 1, pp. 42–47, Feb. 2011. [Online]. Available: http://www.gnu.org/s/parallel
[76] J. Baert. (2018). *Libmorton: C++ Morton Encoding/Decoding Library*. [Online]. Available: https://github.com/Forceflow/libmorton
[77] L. Chenyang, Z. Hong, and W. Nengchao, "Fast N-dimensional Hilbert mapping algorithm," in *Proc. Int. Conf. Comput. Sci. Appl.*, Jun. 2008, pp. 507–513.
[78] A. M. Partl. 2013. *Library for N-Dimensional Hilbert Mapping*. AIP E-science. [Online]. Available: https://github.com/aipescience/libhilbert

CONRAD J. HAUPT received the B.Sc. degree (Hons.) in electrical engineering (information) from the University of the Witwatersrand, Johannesburg, South Africa, in 2017, where he is currently pursuing the M.Sc. degree in electrical engineering on digital data compression.

From 2015 to 2017, he worked as an Engineering Intern for the Square Kilometre Array in South Africa on telecommunications and wireless spectrum management projects. During his M.Sc., he has been a part-time lecturer and head tutor for undergraduate electrical engineering courses. He is a member of the WitsQ Quantum Computing Team, University of the Witwatersrand. He has mentored numerous undergraduate teams for competitions in super-computing and data-science hosted by the Council for Scientific and Industrial Research (CSIR), South Africa. His research interests include low-level software optimization, data-intensive computing, and quantum computing.

EKOW J. OTOO (Member, IEEE) received the B.Sc. degree in electrical engineering from the University of Science and Technology, Kumasi, Ghana, in 1975, the Dip.C.S. degree in computer science from the University of Ghana, Legon, Ghana, in 1976, the M.Sc. degree in computer science from the University of New Castle Upon Tyne, U.K., in 1977, and the Ph.D. degree in computer science from McGill University, Montreal, QC, Canada, in 1983.

From 2010 to 2012, he was acting as the Head-of-School with the School of Computer Science, University of the Witwatersrand, South Africa. He was promoted to a Full Professor with the University of the Witwatersrand, in 2016. Having retired in 2019, he remains a Visiting Professor with the School of Electrical and Information Engineering, University of the Witwatersrand, while residing in California, USA. He is a Panel Member and a Reviewer of grant applicants to the National Science Foundation and of early career grant applications to the Office of Science, Department of Energy, USA. His research interests include database management engines, scientific data management, and parallel and distributed computing.

Dr. Otoo was awarded the R&D 100 Award for Technology Advances for work at Lawrence Berkeley Labs on *Fastbit Indexing Method*, in 2008. He has been a Referee for submitted papers of the Very Large Data Bases conference, Conference on Computing Information, and International Conference on Supercomputing. He has also refereed for the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING and IEEE TRANSACTIONS ON DATA AND KNOWLEDGE ENGINEERING, amongst others.

LING CHENG (Senior Member, IEEE) received the B.Eng. degree *(cum laude)* in electronics and information from the Huazhong University of Science and Technology (HUST), in 1995, and the M.Ing. *(cum laude)* and D.Ing. degrees in electrical and electronics from the University of Johannesburg (UJ), in 2005 and 2011, respectively. He joined the University of the Witwatersrand, in 2010, where he was promoted to a Full Professor, in 2019. He has been a visiting professor at five universities and the principal advisor for over 40 full research postgraduate students. He has published more than 100 research articles in journals and conference proceedings. His research interests include telecommunications and artificial intelligence. He was awarded the Chancellor's medals, in 2005 and 2019, and the National Research Foundation rating, in 2014. The IEEE ISPLC 2015 Best Student Paper Award was made to his Ph.D. student in Austin. He is the Vice-Chair of the IEEE South African Information Theory Chapter. He serves as an associate editor of three journals.