`

# PARALLELIZATION OF CCSDS HYPERSPECTRAL IMAGE COMPRESSION USING C++

**TAN LIT CHEZ**

**UNIVERSITI SAINS MALAYSIA**

**2018**

`

# PARALLELIZATION OF CCSDS HYPERSPECTRAL IMAGE COMPRESSION USING C++

**by**

**TAN LIT CHEZ**

**Thesis submitted in partial fulfillment of the**

**requirement for the degree of**

**Bachelor of Engineering (Mechatronic Engineering)**

**JUNE 2018**

# ACKNOWLEDGEMENTS

This thesis is dedicated to everyone who embarks the journey of expanding the collection of knowledge and transcendent passion for continuous improvement in the field of data compression research, especially regarding hyperspectral image compression.

First of all, I would like to express my deepest gratitude to Dr. Nor Rizuan Mat Noor, my thesis advisor and project supervisor, who has introduced the project to me. His helpful opinion and guidance throughout the research are much appreciated. Besides, his constant motivation, invaluable support, and insightful advice have resulted in the completion of this project.

Next, I would like to thank School of Electrical and Electronic Engineering, Universiti Sains Malaysia (USM) for providing the platform for conducting the research. The facilities and guidance provided by the university are the key factors of completion of this research. Furthermore, I would like to extend my thanks to everyone who helped me in this project, their helpful comments and support throughout the research is greatly appreciated and will be forever remembered.

Last but not least, I offer my regards and blessing to my beloved family and friends for their unconditional support and motivation throughout the project.

`

# TABLE OF CONTENTS

`

`

`

## LIST OF FIGURES

`

# LIST OF TABLES

`

# LIST OF ABBREVIATIONS

| | |
|---|---|
| 2D | Two dimensional |
| 3D | Three dimensional |
| API | Application programming interface |
| AVIRIS | Airborne Visible/Infrared Imaging Spectrometer |
| BI | Band-interleaved |
| BSQ | Band-sequential |
| CCSDS | Consultative Committee for Space Data System |
| CCSDS-MHC | Lossless Multispectral and Hyperspectral Image Compression Standard |
| CPU | Central Processing Unit |
| ESA | European Space Agency |
| FL | Fast Lossless |
| GICI | Group on Interactive Coding of Images |
| GPU | Graphical Processing Unit |
| IDE | Integrated Development Environment |
| JAXA | Japan Aerospace Exploration Agency |
| NASA | National Aeronautics and Space Administration |
| OpenMP | Open Multi-Processing |
| RAM | Random-access Memory |
| VS2010 | Visual Studio 2010 |

`

# PARALLELIZATION OF CCSDS HYPERSPECTRAL IMAGE COMPRESSION USING C++

## ABSTRACT

The advent of space technologies eases the collection information from earth surface through remote sensing. However, the bandwidth and storage limitation impose on spaceborne devices have increased the need for data compression technique. As the response, Consultative Committee for Space Data System (CCSDS) have released Lossless Multispectral and Hyperspectral Image Compression standard (CCSDS-MHC) as the standard to losslessly compress the hyperspectral image taken by spaceborne devices. Currently, most implementation of the CCSDS-MHC algorithm utilizes single processor core for the compression process. However, CCSDS-MHC has the potential to operate on multi-core system with the use of parallelization. With the introduction of multi-core processing system on spaceborne satellite, the execution time of the system can be further decreased. In this research, the aim is to design a parallelization algorithm on CCSDS-MHC using Open Multi-Processing (OpenMP), an open-source C++ application programming interface (API). The first step of the research is converting the CCSDS-MHC algorithm into a full program in C++, with both compression and decompression features. Next, the parallelizable section of the algorithm is identified and coded using OpenMP. The algorithm has been parallelized by dividing the bands of the hyperspectral image into several continuous chunks and running them concurrently. The program is then tested in several systems with different number of threads. The execution of parallelized CCSDS-MHC algorithm shows significant speedup for all the system and hyperspectral image tested.

`

# PENYELARIAN TEKNIK PEMAMPATAN DATA IMEJ HIPERSPEKTRAL CCSDS DENGAN C++

## ABSTRAK

Kemajuan dalam teknologi angkasa lepas telah memudahkan pengumpulan maklumat pada permukaan bumi melalui penderiaan jarak jauh (*remote sensing*). Walau bagaimanapun, had jalur lebar dan penyimpanan pada peranti angkasa lepas telah meningkatkan keperluan untuk teknik pemampatan data (*data compression*). Sebagai respons, jawatankuasa konsul untuk sistem data angkasa lepas (CCSDS) telah mengeluarkan piawaian pemampatan imej multispektral dan hiperspektral tanpa kehilangan (CCSDS-MHC) sebagai piawaian dalam pemampatan imej hiperspektral yang ditangkap oleh peranti angkasa lepas. Pada masa kini, kebanyakan perlaksanaan algoritma CCSDS-MHC menggunakan pemproses dengan sebuah inti pemprosesan (*processor core*). Namun demikian, CCSDS-MHC mempunyai potensi untuk dilaksanakan di prosesor dengan pelbagai inti prosesor melalui proses selari (*parallelization*). Dengan menggunakan sistem pelbagai inti prosesor (*multi-core processing system*) pada satelit, masa untuk pelaksanaan proses pemampatan dapat dipercepatkan. Matlamat untuk penyelidikan ini ialah mereka teknik proses selari dengan menggunakan Open Multi-Processing (OpenMP), sebuah antara muka pengaturcaraan aplikasi (API) sumber terbuka dalam bahasa pengaturcaraan C++. Langkah pertama dalam penyelidikan ini ialah menterjemahkan algoritma CCSDS-MHC kepada program C++, lengkap dengan kebolehan pemampatan dan pemulihan pemampatan. Seterusnya, tempat yang boleh diselarikan dalam algoritma telah dikenalpastikan dan dipengaturcarakan dengan mengunakan OpenMP. Algoritma CCSDS-MHC telah diselarikan dengan membahagikan jalur-jalur imej hiperspektral kepada beberapa potongan yang bersinambung dan dilaksanakan secara serentak. Algoritma yang diselarikan ini telah diuji di beberapa sistem komputer dengan nombor benang (*threads*) yang tidak sama. Keputusan pengujian menunjukkan bahawa algoritma CCSDS-MHC yang diselarikan mampu mencapai peningkatan kelajuan (*speedup*) yang signifikan dalam kesemua sistem konputer dan imej hiperspektral yang digunakan.

`

# CHAPTER 1
# INTRODUCTION

## 1.1 Research Background

Remote sensing, in its broadest sense, can be defined as the method of acquiring the information from an object of interest without physically contacting the object. In this thesis, however, the definition of remote sensing will be strictly defined as the measurement of the object of interest's properties on the earth's surface using data acquired from airborne aircraft and spaceborne satellites. Remote sensing systems, especially satellite-based system, provides many useful applications due to their ability to provide a repetitive and consistent view of the earth. Some of these applications include environmental assessment and monitoring, natural resource exploration, military surveillance and topography mapping [1].

Due to the airborne and spaceborne nature of remote sensing system, the system relies on propagated signals such as electromagnetic waves for information acquisition, the information is then transformed into a two-dimensional spatial grid, which is an image. For most of the recent remote sensing system, the imaging unit consists of the hyperspectral image sensor, which captures few hundreds spectral bands across the visible and infrared region of the electromagnetic spectrum in one run with high spectral resolution. This produces a three-dimensional (3D) image. Due to the high resolution, the image has a very large file size.

Traditionally, the hyperspectral image is retrieved after the airplane carrying the remote sensing system touch down back to earth. However, with the advent of the spaceborne remote sensing system, which provides a better solution for an uninterrupted, real-time surveillance, required the acquired image to be downloaded by the operators. A large file size provides a constant challenge for the operators due to limited onboard storage and limited downlink bandwidth on the satellites.

To overcome the limitation, the hyperspectral image data undergoes compression just after acquisition occurs. There is two type of image compression algorithm, lossless and lossy [2]. The lossless image compression able to produce zero data loss of information during the compression process. Once the data is received by the receiver, the image can be decompressed and reconstructed back to original image perfectly. This method is preferred for satellite-borne application due to the need of post-processing at high fidelity and resolution [3]. There are several algorithms to achieve the lossless compression. One of these algorithm techniques has been released by Consultative Committee for Space Data System (CCSDS), titled CCSDS 123.0-B-1 Lossless Multispectral and Hyperspectral Image Compression standard (Referred as CCSDS-MHC throughout the thesis) [4].

CCSDS-MHC provides state-of-the-art compression performance in low complexity domain, hence it is the most suitable algorithm for space environment among an extensive range of lossless image compression algorithm [5]. However, as the technology becomes more and more sophisticated, the hyperspectral image quality, and hence the image size has also increased. This, in turn, requesting better results in various performance such as execution time.

One of the solution to improve execution time is to undergo parallelization. Parallelization involves dividing a program into several parts that can be solved at the same time with the same result. In fact, there are increasing interest in introducing multicore processors for next-generation satellite [6] which could make use of parallelization for the various task for the satellite. Simultaneously, it is found that CCSDS-MHC have the potential to be parallelized by using multicore processors [7] and a guideline for parallelizing the lossless compression algorithm is provided [8].

Open Multi-Processing (OpenMP) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C++. OpenMP consists of a set of compiler directives and a library of support function for program parallelization. The usage of OpenMP enables the total execution time of CCSDS-MHC to be reduced.

Currently. the open source implementation of CCSDS-MHC is written in Java language. Hence, there is a need to prepare the CCSDS-MHC in C++ language before attempting to add in OpenMP directives. This, in turn, makes this thesis consists of two stages, which the first stage is converting CCSDS-MHC into C++ language and the second stage is to speed up the total execution time of CCSDS-MHC with the usage of parallelization through OpenMP.

## 1.2 Problem Statement

Advancement in optical imager for remote sensing system have increases the image resolutions gathered from the system, which increases the size of hyperspectral data generated from the system. This demands a better compression algorithm to enable the data generated transmitted efficiently back to the operator. As a response, CCSDS has release CCSDS-MHC, a predictive lossless compression algorithm as the compression standard for hyperspectral image [4]. Nevertheless, there is active research on improving the performance of the algorithm further. Hopson et al. [7] have demonstrated the potential of CCSDS-MHC to be parallelized by using multicore processors with OpenMP. Olaru et al. [8] have provided a guideline for parallelizing the lossless compression algorithm based on master-worker design paradigm. Furthermore, Davidson et al. [5] have developed an implementation of parallelization technique under CCSDS-MHC using modified

CUDA on multispectral images. Then, Schwartz et al. [9] show improvement in processing time and energy consumption by segmenting the input images into tiles. These works show a lot of promise in developing the parallelization technique to sped up CCSDS-MHC algorithm. In this research, we are trying to understand the CCSDS-MHC compression and decompression algorithm and attempt to improve its performance in term of total execution time by using parallelization method through OpenMP API.

### 1.3 Research Objective

1. To develop Lossless Multispectral and Hyperspectral Image Compression (CCSDS-MHC) decompression algorithm in C++ language.
2. To improve CCSDS-MHC algorithm's speed through parallel processing by using OpenMP application programming interface (API).

### 1.4 Scope of Research

This research consists of two parts. The first part is to develop CCSDS-MHC decompression algorithm in C++ language with the reference of an open source CCSDS-MHC algorithm written in Java code. The second part is to find and develop the parts for parallelization within CCSDS-MHC algorithm with the aid of OpenMP API. Both parts are developed under Visual Studio 2010 (VS 2010) as the main platform.

The mode for the CCSDS-MHC algorithm is fixed throughout the research. The algorithm is set to be in full prediction mode, with neighbor-oriented local sum type and sample-adaptive entropy coder [10]. Any evaluation of the algorithm will be based on this mode.

Two sets of four hyperspectral images are used for result evaluation, which is Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) and Hyperion. All AVIRIS images are cropped into image size of 512 lines × 512 samples × 224 bands and all Hyperion images are cropped into image size of 256 lines × 256 samples × 196 bands. While original Hyperion images have 242 bands, only 196 bands which were classified as unique and calibrated, are selected [22]. All images are encoded in band-sequential order (BSQ) format. Each image pixel consists of 16-bits data.

Three individual computers will be used to evaluate the result. These computers have different number of cores to evaluate the performance of the algorithm when running in parallel mode compared to sequential (without parallelization) mode. The execution time of the compression part served as the sole performance indicator in this thesis. The parallelization of decompression part is not concerned in this thesis as this process is executed on the ground rather than on-board of satellites.

## 1.5 Thesis Outline

This thesis consists of five chapters.

Chapter 1 introduces the research with the background and importance of the research. This chapters also highlights the problem statements, objective and the scope of the research.

Chapter 2 gives the brief introduction of the CCSDS-MHC algorithm and the fundamental concepts of the topic involved the hyperspectral imaging and parallelization concepts. This chapter also summaries the literature reviews that have been done on the parallelization of the CCSDS-MHC algorithm.

Chapter 3 gives the methodology for the development of CCSDS-MHC algorithm in C++ language and development of parallelization of the algorithm. This chapter also explains the method to evaluate the performance of the algorithm in terms of losslessness and execution time.

Chapter 4 shows the result of the evaluation of CCSDS-MHC algorithm performance. This chapter also discusses any significance obtained from the result.

Chapter 5 concludes the research study. Any possible improvements and recommendations for future research are discussed in this chapter.

`

# CHAPTER 2
# LITERATURE REVIEW

## 2.1 Overview

In this chapter, the fundamentals of CCSDS Lossless Multispectral and Hyperspectral Image Compression algorithm (CCSDS-MHC) will be explained. The first part (Section 2.2) explains some fundamental terms and concepts in hyperspectral image compression. The second part (Section 2.3) explains the CCSDS-MHC compression algorithm. The third part (Section 2.4) explains the CCSDS-MHC decompression algorithm. The fourth part (Section 2.5) explains some fundamental terms and concepts in parallel programming and OpenMP. The fifth part (Section 2.6) discusses current academic effort in developing CCSDS-MHC parallelization algorithm. The chapter ends with a summary (Section 2.7).

## 2.2 Hyperspectral Image Compression

Several fundamental terms and concepts of hyperspectral image compression will be explained in this section. The section starts with the introduction of a hyperspectral image (Subsection 2.2.1), followed by the image redundancies that can be used to compress an image (Subsection 2.2.2). The model of a lossless hyperspectral image compression system will be revealed at subsection 2.2.3).

`

### 2.2.1 Hyperspectral Image

Hyperspectral imaging is commonly defined as ‘the simultaneous acquisition of spatially co-registered images, in many narrow, spectrally contiguous bands, measured in calibrated radiance units, from a remotely operated platform’ [12]. The hyperspectral image is produced when a hyperspectral sensor in a hyperspectral remote sensing system acquire the images throughout the visible and infrared bands of the electromagnetic spectrum [13]. For example, AVIRIS, NASA’s airborne hyperspectral sensor, acquire images from 224 contiguous spectral bands ranging from 0.4 μm to 2.5 μm. As a result, a hyperspectral image is a three dimensional (3-D) images, with two spatial components (width and height) and one spectral component (band). Figure 2-1 shows the structure of a hyperspectral image with $N_x \times N_y$ spatial size and $N_z$ bands.

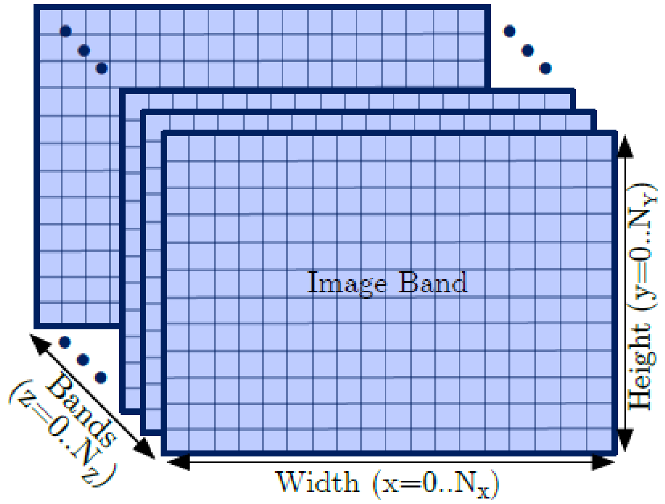
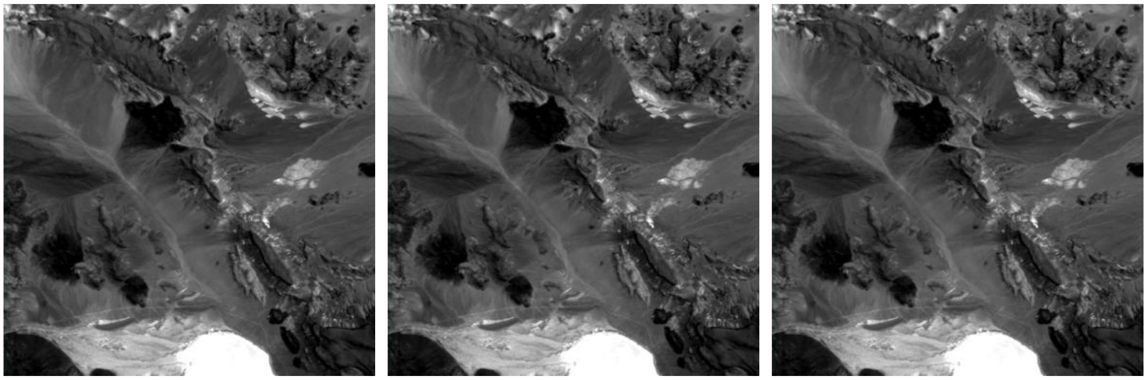


Figure 2-1: Structure of Hyperspectral Image [3]

### 2.2.2 Image Redundancies

An image, generally, can be compressed by eliminating some of the redundancies exists within the image. For any 3-D hyperspectral image, there are 4 categories of redundancy, which is statistical, spatial, human vision and spectral [14]. Statistical redundancy is associated with the representation of image pixels with code. Spatial redundancy is the phenomenon where the information in one pixel can be inferred from neighboring pixels. Human vision redundancy is based on the fact where human vision system is not sensitives to high-frequency contents. Finally, spectral redundancy is the phenomenon of a high degree of correlation between the bands, i.e. one band contains nearly the same information as the adjacent bands. Figure 2-2 shows the example of spectral redundancy in hyperspectral images, the information between the bands is virtually similar.



(a) 30th Band (b) 31st Band (c) 32nd Band

Figure 2-2: AVIRIS Lunar Scene 2 from the 30th to 32nd Bands

### 2.2.3 Lossless Hyperspectral Image Compression

For most of the space application, it is desirable for no data loss during compression and the original data can be recovered exactly from compressed data, this is known as lossless compression. Lossless compression reduces all redundancies except human vision redundancies. A typical lossless hyperspectral image compression system consists of two blocks, which is spectral decorrelation and spatial compression. Spectral decorrelation deal with spectral redundancies while spatial compression deal with spatial and statistical redundancies. Figure 2-3 shows a typical model of lossless hyperspectral image compression. Some of the performance indicators of an image compression system are compression ratio and execution time.

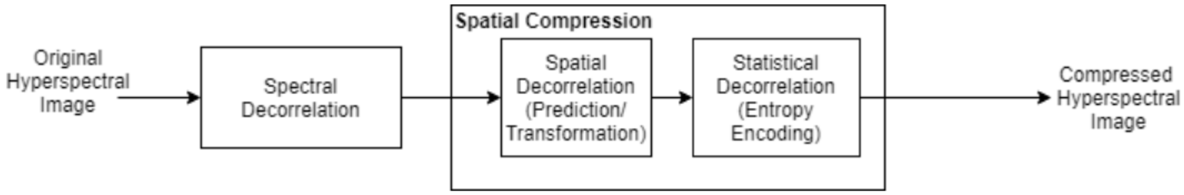

Spatial Compression
Original Hyperspectral Image
Spectral Decorrelation
Spatial Decorrelation (Prediction/ Transformation)
Statistical Decorrelation (Entropy Encoding)
Compressed Hyperspectral Image

Figure 2-3: Typical model of Lossless Hyperspectral Image Compression

## 2.3 CCSDS Lossless Multispectral and Hyperspectral Image Compression

The Consultative Committee for Space Data System, or CCSDS, is a multi-national forum for the development of communications and data handling standards for spaceflight. It is founded in 1982 by the major space agencies of the world, such as National Aeronautics and Space Administration (NASA), Japan Aerospace Exploration Agency (JAXA) and European Space Agency (ESA). The algorithm and protocol released by CCSDS are frequently implemented in space-related image and data processing literature, with applications such as imaging spectrometer [15], space communications [16] and solar corona satellites [17]. On May 2012, CCSDS have released CCSDS 123.0-B-1, or CCSDS Lossless Multispectral and Hyperspectral Image Compression (CCSDS-MHC), as the recommended standard for hyperspectral image compression [4]. The algorithm is a variation of Fast Lossless (FL) algorithm proposed by Klimish[18], which is a 3D compression technique which is that have better overall results compare to other 2D approaches and 2D algorithm [3].

### 2.3.1 General Overview

CCSDS-MHC is a data compressor which is able to take in multispectral and hyperspectral images as input. An *image*, which will be defined in section 2.3.2, is a 3D array of signed or unsigned integer sample value. The difference between multispectral and hyperspectral images, on the other hand, is vague. Some literature sources differentiate the terms with a number of bands ranging from 10 [12] to 100 [13], with a higher number indicating it is hyperspectral images, while some sources take account of their spectral resolution [19].

The output of the compressor is an encoded bit stream from which the input image can be recovered exactly. The size of the output is not fixed since the length of compressed varies with image due to variation in image content.

The compressor consists of two parts, namely predictor and encoder. Figure 2-4 shows the schematic of CCSDS-MHC compressor.

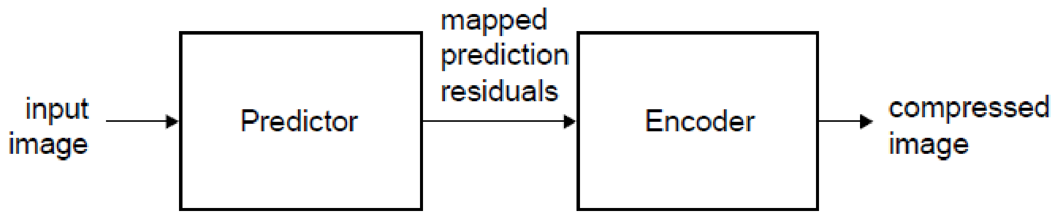


Figure 2-4: Compressor Schematic [4]

The predictor, which will be covered in section 2.3.3, predicts the value of each image sample based on the values of nearby sample in a small 3D neighborhood. The difference between the predicted and actual sample value, or prediction residual is the output of the predictor. It is a mapped to an unsigned integer which has same number of bits as the input data sample.

`

The encoder, which will be covered in section 2.3.4, encode the mapped prediction residuals losslessly to form a header followed by the body. The header contains compression parameters of the process. The body contains codewords, which are coded mapped prediction residuals.

### 2.3.2 Definition of Image

An image (Figure 2-5) is defined as a 3D array of signed or unsigned integer sample values, $s_{z,y,x}$, where x and y are the two spatial dimensions (width, $N_X$ and height $N_Y$), while z indicates the spectral dimension (band, $N_Z$). Additionally, each sample will be coded in D bits, where D is the *dynamic range*, with range of 2 ≤ D ≤ 16. Since all the images that used in this thesis is taken with 16-bit color depth, it will have D value equal to 16.

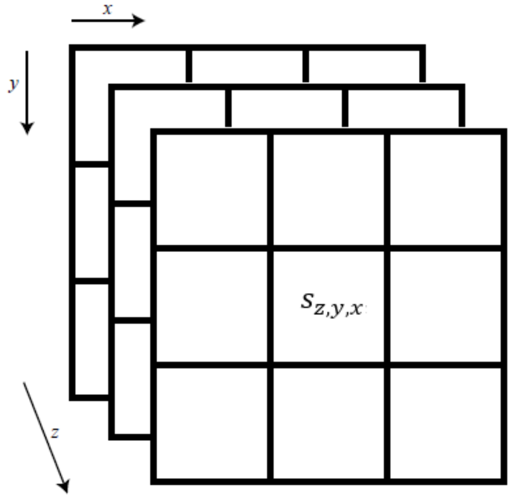


Figure 2-5: 3D Array Representation of an Image

*Notes: For notational simplicity, the symbol such as* $s_{z,y,x}$ *will be denoted as* $s_z(t)$ *throughout the paper, where* $t = y \cdot N_X + x$

`

### 2.3.3 Predictor

The function of the predictor is to calculate *predicted sample values,* $\hat{s}_{z,y,x}$and *mapped prediction residual*, $\delta_{z,y,x}$ from each input image sample, $s_{z,y,x}$. The flowchart of the predicting process is depicted as Figure 2-6. The flowchart also includes the section number of each sub processes.

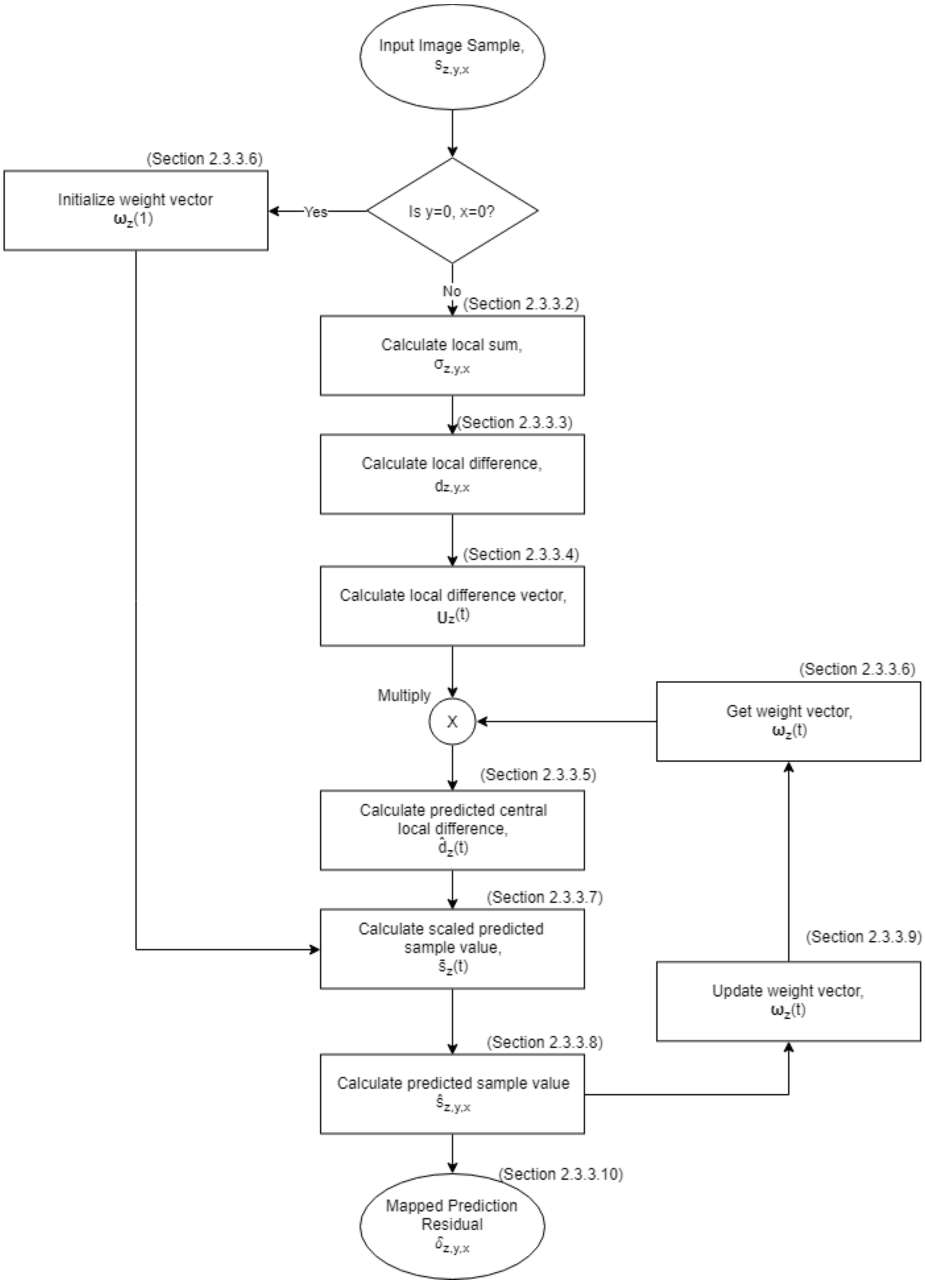

Input Image Sample, $s_{z,y,x}$
Is y=0, x=0?
Yes
No
(Section 2.3.3.6)
Initialize weight vector $\omega_z(1)$
(Section 2.3.3.2)
Calculate local sum, $\sigma_{z,y,x}$
(Section 2.3.3.3)
Calculate local difference, $d_{z,y,x}$
(Section 2.3.3.4)
Calculate local difference vector, $U_z(t)$
Multiply
X
(Section 2.3.3.6)
Get weight vector, $\omega_z(t)$
(Section 2.3.3.5)
Calculate predicted central local difference, $\hat{d}_z(t)$
(Section 2.3.3.7)
Calculate scaled predicted sample value, $\tilde{s}_z(t)$
(Section 2.3.3.9)
Update weight vector, $\omega_z(t)$
(Section 2.3.3.8)
Calculate predicted sample value $\hat{s}_{z,y,x}$
(Section 2.3.3.10)
Mapped Prediction Residual $\delta_{z,y,x}$

Figure 2-6: Flowchart of Prediction Process

`

### 2.3.3.1 Prediction Neighborhood

Based on Figure 2-6, the prediction process started with finding the local sum (Section 2.3.3.2) of a sample $s_{z,y,x}$. This required a defined 3D neighborhood known as *prediction neighborhood*. Figure 2.7 shows a typical prediction neighborhood. The prediction neighborhood contains several nearby sample of current spectral band and P previous bands, where P is known as *number of bands for prediction*. P is defined by the user and is a positive integer not more than 15. In short, the prediction neighborhood comprises values from band z, i.e. current band until band $(z - P_z^*)$, where $P_z^*$ is defined in equation (2.1).

$$P_z^* = \min(z, P) \tag{2.1}$$

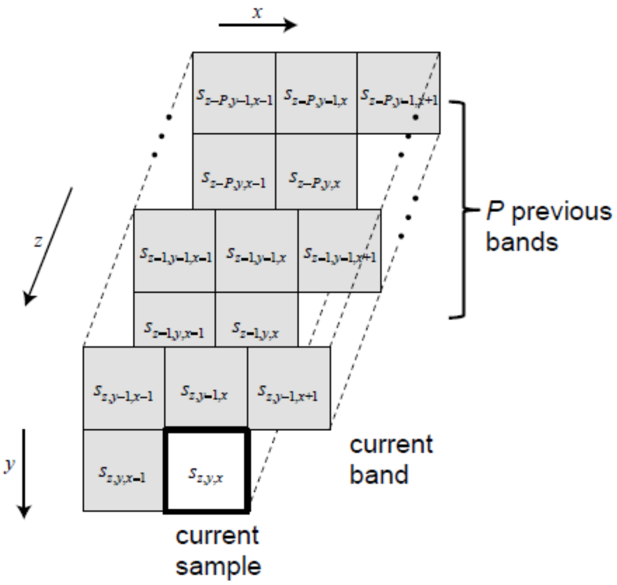
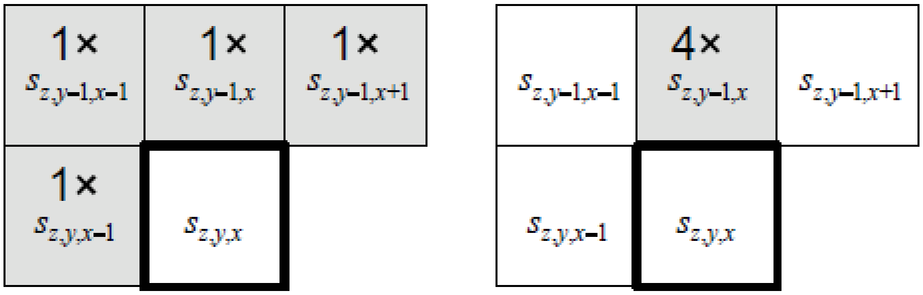


Figure 2-7: Typical Prediction Neighborhood [4]

`

### 2.3.3.2 Local Sum

The local sum, $\sigma_{z,y,x}$ for any sample $s_{z,y,x}$ is the sum of all sample inside the prediction neighborhood. The user may choose one of two methods to calculate the local sum, which is neighbor-oriented method or column-oriented method. Figure 2-8 illustrate the both method.



a) Neighbor-oriented method b) Column-orientated method

Figure 2-8: Samples Used to Calculate Local Sums

Mathematically, the neighbor-oriented local sum is defined as in equation (2.2).

$$\sigma_{z,y,x} = \begin{cases} \mathrm{S}_{z,y-1,x-1} + \mathrm{S}_{z,y,x-1} + \mathrm{S}_{z,y-1,x} + \mathrm{S}_{z,y-1,x+1} & y > 0, 0 < x < N_x - 1 \\ 4(\mathrm{S}_{z,y,x-1}) & y = 0, x > 0 \\ 2(\mathrm{S}_{z,y-1,x} + \mathrm{S}_{z,y-1,x+1}) & y > 0, x = 0 \\ \mathrm{S}_{z,y-1,x-1} + \mathrm{S}_{z,y,x-1} + 2(\mathrm{S}_{z,y-1,x}) & y > 0, x = N_x - 1 \end{cases} \qquad (2.2)$$

Similarly, the column-orientated local sum is defined as in equation (2.3).

$$\sigma_{z,y,x} = \begin{cases} 4\ (\mathrm{S}_{z,y-1,x}) & y > 0 \\ 4\ (\mathrm{S}_{z,y,x-1}) & y = 0, x > 0 \end{cases} \qquad (2.3)$$

`

### 2.3.3.3 Local Difference

Based on Figure 2-6, the local difference of each sample $s_{z,y,x}$ is calculated based on the local sums. For each spectral band, there are one *central local difference*, $d_{z,y,x}$ and three *directional local differences*, $d^{N}_{z,y,x}$, $d^{W}_{z,y,x}$, $d^{NW}_{z,y,x}$. Figure 2.9 shows the four local differences involved

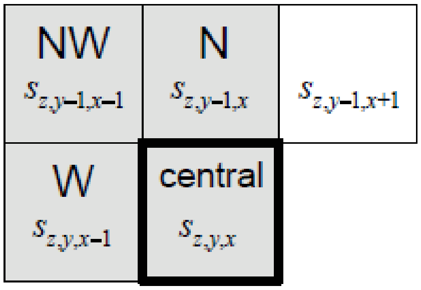


Figure 2-9: Computing Local Differences in a Spectral Band

Mathematically, when x and y are not both zero, these local difference is defined by equation (2.4) – (2.7).

$$\mathrm{d}_{z,y,x} = 4\mathrm{S}_{z,y,x} - \sigma_{z,y,x} \qquad (2.4)$$

$$\mathrm{d}^{\mathrm{N}}_{z,y,x} = \begin{cases} 4\,(\mathrm{S}_{z,y-1,x}) - \sigma_{z,y,x} & y > 0 \\ 0 & y = 0 \end{cases} \qquad (2.5)$$

$$\mathrm{d}^{\mathrm{W}}_{z,y,x} = \begin{cases} 4\left(\mathrm{S}_{z,y,x-1}\right) - \sigma_{z,y,x} & y > 0, x > 0 \\ 4\,(\mathrm{S}_{z,y-1,x}) - \sigma_{z,y,x} & y > 0, x = 0 \\ 0 & y = 0 \end{cases} \qquad (2.6)$$

$$\mathrm{d}^{\mathrm{NW}}_{z,y,x} = \begin{cases} 4\left(\mathrm{S}_{z,y-1,x-1}\right) - \sigma_{z,y,x} & y > 0, x > 0 \\ 4\,(\mathrm{S}_{z,y-1,x}) - \sigma_{z,y,x} & y > 0, x = 0 \\ 0 & y = 0 \end{cases} \qquad (2.7)$$

`

### 2.3.3.4 Local Difference Vector

The next step involves the calculation of local difference vector, $\boldsymbol{U}_z(t)$. The local difference vector is valid when x and y are not both zero. The user may choose one of the two modes to decide the contents of local difference vector, which are *reduced prediction mode* and *full prediction mode*. Under reduced prediction mode, only central local differences, $d_{z,y,x}$ from preceding $P_z^*$ bands will be considered. Mathematically, this is presented by equation (2.8).

$$\boldsymbol{U}_z(t) = \begin{bmatrix} d_{z-1}(t) \\ d_{z-2}(t) \\ \vdots \\ d_{z-P_z{}^*}(t) \end{bmatrix} \tag{2.8}$$

Under full prediction mode, both central local differences, $d_{z,y,x}$ from preceding $P_z^*$ bands and three direction local differences from current bands are taken into account in local difference vector. Mathematically, this is represented by equation (2.9)

$$\boldsymbol{U}_z(t) = \begin{bmatrix} \mathrm{d}_\mathrm{z}^\mathrm{N}(t) \\ \mathrm{d}_\mathrm{z}^\mathrm{W}(t) \\ \mathrm{d}_\mathrm{z}^\mathrm{NW}(t) \\ d_{z-1}(t) \\ d_{z-2}(t) \\ \vdots \\ d_{z-P_z{}^*}(t) \end{bmatrix} \tag{2.9}$$

Due to the presence of two mode, there are different number of elements inside the vector, these are denoted by using symbol $C_z$, where in reduced prediction mode, $C_z = P_z^*$ and in full prediction mode, $C_z = P_z^* + 3$.

### 2.3.3.5 Predicted Central Local Difference

Next, the local difference vector, $\boldsymbol{U}_z(t)$ is multiplied with the weight vector $\boldsymbol{W}_z(t)$ to produce a single-valued predicted central local difference, $\hat{d}_z(t)$. Mathematically, it is represented as equation (2.10)

$$\hat{d}_z(t) = \ \boldsymbol{W}_z{}^T(t) \ \ \cdot \boldsymbol{U}_z(t) \tag{2.10}$$

### 2.3.3.6 Weight Vector

The weight vector, $\boldsymbol{W}_z(t)$ which contains $C_z$ elements depends on the prediction mode. Mathematically, under reduced prediction mode, the weight vector is defined by equation (2.11)

$$\boldsymbol{W}_z{}^T(t) \ = \begin{bmatrix} \omega_{\mathrm{z}}^{(1)}(t) \\ \omega_{\mathrm{z}}^{(2)}(t) \\ \vdots \\ \omega_{\mathrm{z}}^{(\,P_z{}^*)}(t) \end{bmatrix} \tag{2.11}$$

Similarly, under full prediction mode, the weight vector is defined by equation (2.12)

$$\boldsymbol{W}_z{}^T(t) \ = \begin{bmatrix} \omega_{\mathrm{z}}^{\mathrm{N}}(t) \\ \omega_{\mathrm{z}}^{\mathrm{W}}(t) \\ \omega_{\mathrm{z}}^{\mathrm{NW}}(t) \\ \omega_{\mathrm{z}}^{(1)}(t) \\ \omega_{\mathrm{z}}^{(2)}(t) \\ \vdots \\ \omega_{\mathrm{z}}^{(\,P_z{}^*)}(t) \end{bmatrix} \tag{2.12}$$

The weight values $\omega_z^N(t), \dots, \omega_z^{(1)}(t), \dots, \omega_z^{(P_z^*)}(t)$ etc. can be initialized with default values or customized values by user. The default weight values are given by equation (2.13), (2.14) and (2.15)

$$\boldsymbol{\omega}_z{}^N(1) \ = \boldsymbol{\omega}_z{}^W(1) \ = \boldsymbol{\omega}_z{}^{NW}(1) \ = 0 \tag{2.13}$$

$$\boldsymbol{\omega}_z{}^{(1)}(1) \ = \frac{7}{8} \cdot 2^{\Omega} \tag{2.14}$$

$$\boldsymbol{\omega}_z{}^{(i)}(1) \ = \left\lfloor \frac{1}{8} \cdot \boldsymbol{\omega}_z{}^{(i-1)}(1) \right\rfloor, i = 2,3, \dots, P_z^* \tag{2.15}$$

Where Ω is a user-defined value known as *weight resolution value*, which is an integer with a range of 4 to 19. The value of weight values will be capped with a minimum value $\omega_{min}$ and a maximum value $\omega_{max}$. Both values are defined in equation (2.16) and (2.17)

$$\omega_{min} = \ -2^{\Omega+2} \tag{2.16}$$

$$\omega_{max} = \ \ 2^{\Omega+2} - 1 \tag{2.17}$$

### 2.3.3.7 Scaled Predicted Sample Value

The predicted central local difference, $\hat{d}_z(t)$ is used to calculate scaled predicted sample value, $\tilde{s}_z(t)$. Mathematically, $\tilde{s}_z(t)$ is defined as in equation (2.18)

$$\tilde{S}_{z,y,x} = \begin{cases} clip\left(\begin{array}{c}\left[\dfrac{mod_R^*\left[\hat{d}_z(t) + \ 2^{\Omega}(\sigma_z(t) - 4S_{mid})\right]}{2^{\Omega+1}}\right] \\ +2S_{mid} + 1, \{2S_{mid}, 2S_{max} + 1\}\end{array}\right) & t > 0 \\ 2S_{z-1}(t) & t = 0, P > 0, z > 0 \\ 2S_{mid} & t = 0 \ and \ (\ P = 0 \ or \ Z = 0\ ) \end{cases} \tag{2.18}$$

`

Where,

a) Clip function is defined by equation (2.19)

$$clip\ (X, \{X_{min}, X_{max}\}) = \begin{cases} X_{min} & \mathrm{X} < X_{min} \\ X & X_{min} \leq X \leq X_{max} \\ X_{max} & \mathrm{X} > X_{max} \end{cases} \qquad (2.19)$$

b) R is a user-selected parameter known as *register size parameter*, which is an integer in the range $\max\{32, D + \Omega + 2\} \leq R \leq 64$. The mod$_R$ function in then defined by equation (2.20)

$$mod_R^*[X] = \left((x + 2^{R-1})\ mod\ 2^R\right) - 2^{R-1} \qquad (2.20)$$

Where mod(x) is the modulo of x

c) $s_{min}$, $s_{max}$, $s_{mid}$ is lower sample value limit, the upper sample value limit, and a mid-range sample value, respectively. The value of these constants is related to type of integer of the sample and dynamic range, D.

For sample with unsigned integer,

$$s_{min} = 0,\ s_{max} = 2^D - 1,\ s_{mid} = 2^{D-1} \qquad (2.21)$$

For sample with signed integer,

$$s_{min} = -2^{D-1},\ s_{max} = 2^{D-1} - 1,\ s_{mid} = 0 \qquad (2.22)$$

### 2.3.3.8 Predicted Sample Value

The predicted sample value $\hat{s}_{z,y,x} \equiv \hat{s}_z(t)$ is then given by equation (2.23)

$$\hat{s}_z(t) = \left\lfloor \frac{\tilde{s}_z(t)}{2} \right\rfloor \qquad (2.23)$$

### 2.3.3.9 Weight Update

Before finding mapped prediction residual, $\delta_{z,y,x}$, there is a need to update the weight vector, $\boldsymbol{W}_z(t)$. The weight update required two variable, which is scaled prediction error, $e_z(t)$ and weight update scaling exponent, $\rho(t)$.

The scaled prediction error, $e_z(t)$ is the difference between twice the sample value minus the scaled predicted sample value, $\tilde{s}_z(t)$. Mathematically, it is given by equation (2.24).

$$e_z(t) = 2s_z(t) - \tilde{s}_z(t) \tag{2.24}$$

For t >0 (both x and y is not equal to zero), weight update scaling exponent, $\rho(t)$ is defined as equation (2.25)

$$\rho\,(t) \;=\; clip\,\left(v_{min} + \left[\frac{t - N_x}{t_{inc}}\right], \{v_{min}, v_{max}\}\right) + D - \Omega \tag{2.25}$$

Where $v_{min}$, $v_{max}$ and $t_{inc}$ (weight update factor change interval) are user-defined integer parameters with following constraints:

a) $-6 \leq v_{min} \leq v_{max} \leq 9$.

b) $2^4 \leq t_{inc} \leq 2^{11}$

The weight update can be thus calculated with the equation (2.26)

$$W_z(t+1) \tag{2.26}$$

$$= \;clip\,\left(W_z(t) \;+\; \left[\frac{1}{2}\left(sgn^{+}[e_z(t)] \cdot 2^{-\rho(t)} \cdot U_z(t) + 1\right)\right], \{\omega_{min}, \omega_{max}\}\right)$$

### 2.3.3.10 Mapped Prediction Residual

Finally, the mapped prediction residual, $\delta_z(t)$ can be defined as equation (2.27)

$$\delta_{z,y,x} = \begin{cases} |\Delta_z(t)| + \theta_z(t) & |\Delta_z(t)| > \theta_z(t) \\ 2|\Delta_z(t)| & 0 \leq (-1)^{s_z(t)}\Delta_z(t) \leq \theta_z(t) \\ 2|\Delta_z(t)| - 1 & \text{otherwise} \end{cases} \qquad (2.27)$$

Where the prediction residual, $\Delta_z(t)$ is the difference between the predicted and actual sample values, mathematically it is defined in equation (2.28).

$$\Delta_z(t) = s_z(t) - \hat{s}_z(t) \qquad (2.28)$$

and $\theta_z(t)$ is defined as equation (2.29).

$$\theta_z(t) = \min\{\hat{s}_z(t) - s_{min}, s_{max} - \hat{s}_z(t)\} \qquad (2.29)$$

### 2.3.4 Encoder

The encoder creates a compressed image. It consists of a *header* followed by a *body*. The header contains image and compression parameters while the body contains mapped prediction residual coded as codewords.

The structure of the compressed image is shown in Figure 2-10.

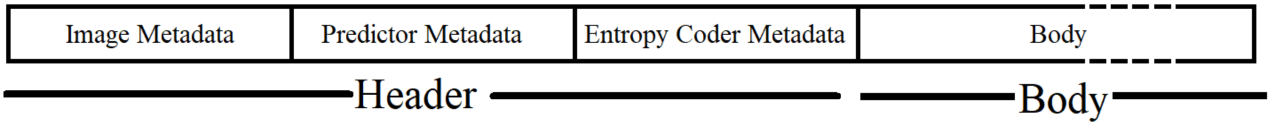


Figure 2-10: Structure of Compressed Image

There is two approaches to encode the body, which is *sample-adaptive entropy coding* and *block-adaptive entropy coding*.

Under sample adaptive entropy coding, mapped prediction residuals are encoded using variable-length binary codewords. The codes used are selected based on the value of *adaptive code selection statistics.* The statistics will be updated after each sample is encoded within the same band, z. Each band has separate statistics. The compressed image size does not depend on the encoding order used.

Under block-adaptive entropy coding, a sequence of mapped prediction residuals is partitioned into short blocks. The encoding method used is independently and adaptively selected for each block. Since the mapped residuals in a block may form differently depending on the encoding order, the compressed image size depends on the encoding order.

Nevertheless, in this thesis, only the algorithm of sample-adaptive entropy coding will be discussed.

In short, the whole process of sample-adaptive entropy coder can be summarized as the flowchart in Figure 2-11.

`

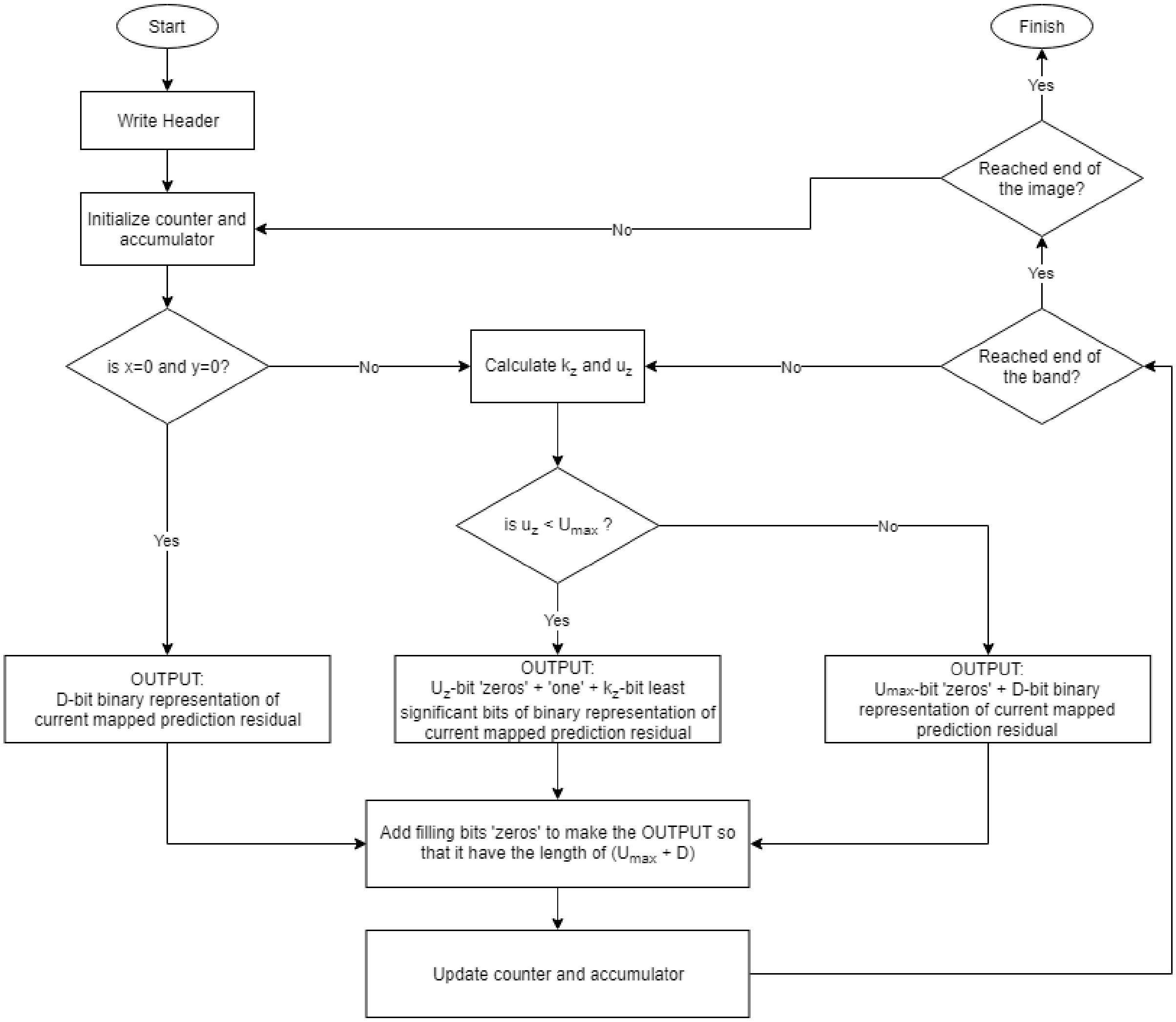

Start
Write Header
Initialize counter and accumulator
is x=0 and y=0?
No
Yes
Calculate $k_z$ and $u_z$
is $u_z$ < $U_{max}$ ?
No
Yes
OUTPUT: D-bit binary representation of current mapped prediction residual
OUTPUT: $U_z$-bit 'zeros' + 'one' + $k_z$-bit least significant bits of binary representation of current mapped prediction residual
OUTPUT: $U_{max}$-bit 'zeros' + D-bit binary representation of current mapped prediction residual
Add filling bits 'zeros' to make the OUTPUT so that it have the length of ($U_{max}$ + D)
Update counter and accumulator
Reached end of the band?
No
Yes
Reached end of the image?
No
Yes
Finish

Figure 2-11: Flowchart of Sample-Adaptive Entropy Coder

### 2.3.4.1 Encoder Header

Based on Figure 2-10, the first portion of the compressed image is the header, it contains 3 parts, *image metadata* (Section 2.3.4.2), *predictor metadata* (Section 2.3.4.3) and *entropy coder metadata* (Section 2.3.4.4).