CCSDS-MHC on Raspberry Pi for Lossless Hyperspectral Image Compression

N A A Samah¹, N R M Noor¹*, E A Bakar² and M K M Desa¹

¹School of Electrical & Electronic Engineering, Universiti Sains Malaysia, Engineering Campus, 14300, Penang, Malaysia
²School of Aerospace Engineering, Universiti Sains Malaysia, Engineering Campus, 14300, Penang, Malaysia

Email: nrnm@usm.my

Abstract. This paper is about Consultative Committee for Space Data System for Lossless Multispectral and Hyperspectral Image Compression (CCSDS-MHC) algorithm that is implemented on Raspberry Pi 3 Model B+ system using Open Multi-Processing (OpenMP). CCSDS-MHC algorithm along with the full prediction mode is opted due to its best compression ratio (CR) performance. The issue of Hyperspectral Image Compression is the loss of data when compress, thus CCSDS algorithm is used in this research. Besides, with current technologies that require low-power but high-performance devices, Raspberry Pi was chosen to be tested in terms of its performances while being compared to other platforms. AVIRIS (airborne) and Hyperion (spaceborne) are used to test the performance of the system. OpenMP is introduced to simplify the computational operation through parallelization to take advantage of the multi-core architecture of the hardware system. In term of execution time, CCSDS-MHC algorithm when parallelize using OpenMP gave the best performances about 69.4% for AVIRIS 1997 dataset, 69.3% for AVIRIS 2006 dataset and 67.7% for Hyperion dataset. The execution time of performance CCSDS-MHC in Raspberry Pi 3 Model B+ comparing with other different multicore platform is validate and the mean parallelize is measured. The result of comparison proves that faster performing task gives the best result due to higher speed of CPU-cores performances. From this research, it is proven that Raspberry Pi which is a low-power embedded platform able to compress the hyperspectral images at optimized speed with the implementation of OpenMP through CCSDS algorithm. Therefore, this research will be useful for further studies in lossless of hyperspectral images.

1. Introduction

In this era, our planet is continuously being observed by airborne and satellite sensors that acquires tons of data to be processed and analyzed daily. One of the types of remote sensing is hyperspectral sensing, also referred to as hyperspectral imaging. Hyperspectral imaging has been widely employed in real life applications industrial, biomedical, and biometric applications such as face detection and recognition [1].

Hyperspectral images are represented in a forms of three-dimensional (x, y, λ) hyperspectral data cube for processing and analysis, where x and y represent two spatial dimensions of the scene, and λ represents the spectral dimension [2]. The hyperspectral cubes are generated from airborne sensors like
NASA's Airborne Visible/Infrared Imaging Spectrometer (AVIRIS), or from satellites like NASA's EO-1 with its hyperspectral instrument Hyperion.

The images need to be transmitted to the ground from the sensors for specific applications. However, the size of these images is typically enormous in quantity due to the bulky number of capturing wavelengths and the length of compressed images that vary from image to image. Thus, the efficient compression techniques are required to fit the available bandwidth for reducing the transmission time [3].

Due to the limitation that impacts the loss of data, there is a need to improve the algorithm in order to compress a lossless image. Therefore, CCSDS-MHC algorithm is developed. The CCSDS-MHC algorithm is a set of reformulations at the prediction stage [4]. However, in this project all parameters of hyperspectral image compression is taken from the papers of international conference of data compression [5] and has been set out into specific declaration.

This research focuses on the lossless compression of AVIRIS images and Hyperion images by using CCSDS-MHC algorithm. Since the aim of the project is a lossless compression of hyperspectral imaging using CCSDS-MHC algorithm, only full predictions will be used.

Dividing the hyperspectral image into blocks and efficiently distributing the prediction and encoding workload to independent tasks running in parallel can improve the compression speed significantly [6]. Therefore, the parallelization of OpenMP yields a large impact to the program execution time [7].

This research is related to lossless hyperspectral image where the need only lies in wanting to compress the image on the satellite which could be time saving with low memory performances that is relatable by the article [8]. Embedded systems are often used in the recent trends. Embedded system has a combination of a computer’s hardware and software function where it is capable of being programmable to be designed in a specific manor or function. Research in [9] proves that FPGA is one of the low-memory and high-performance architectures in the implementation of CCSDS. Graphical Processing Unit (GPU) implementation has been performed such as in [10] but can only be suitable for terrestrial application due to high power consumption of a GPU [11] and low power requirement of satellite-based remote sensing [12]. A compact low-cost and portable spectral imaging system for general purposes through Raspberry Pi has been implemented [13].

Based on the research, Raspberry Pi 3 Model B+ is one of the best low-cost, low-power consumption embedded platforms architecture to implement CCSDS-MHC algorithm for lossless hyperspectral image compression.

2. Methodology
In this research flow, there are nine main part subsystem that combines together to form into one system. The main part for each block in the flowchart Figure 1 will be discussed in detail in the following sections.

The hardware setup consists of four primarily important ports namely micro SD Card port, Ethernet port, HDMI port, and micro USB port that are being used as a power supply. HDMI cable, micro USB cable, Local Area Network (LAN) cable and micro SD Card with 32GB memory were prepared for the connection of the whole hardware setup. Raspberry Pi 3 Model B+ board is used as the main device to implement the CCSDS-MHC algorithm in C language.

For software development, Codeblocks Software is used to operate the CCSDS-MHC algorithm of C language and is run under Raspbian operating system (OS). To establish an interface connection between Mac OS and Raspbian OS, VNC Viewer is used through internet network to control the desktop of the computer in real-time between laptops or another computer as remote control.
In CCSDS-MHC algorithm, hyperspectral image which consist of three-dimensional array of integer is set as input to the compressor in an image. Due to the differences type of image content, the length of compressed images will vary from image to image which means the compressed image is variable-length. Therefore, to compress the hyperspectral images without any loss of information there are two types of functional parts named predictor and encoder as shown in Figure 2.

In the predictor, adaptive linear prediction method is used to predict the value of each image sample that was performed sequentially in a single pass. The prediction residual which is the differences between the predicted and actual sample values will be mapped on to an unsigned integer that can be represented using the same number of bits as the input data sample.

There are two ways to perform the prediction of the neighbouring samples which is by using column-oriented (i.e. reduced-prediction mode) or neighbour-oriented local sums (i.e. full-prediction mode, FP) that can be applied. The use of reduced-prediction mode in a combination with column-oriented local sums tends to yield smaller compressed image data volumes for raw (uncalibrated) input images from push-broom imagers. The use of full-prediction mode in a combination with neighbour-oriented local sums tends to yield smaller compressed image data volumes for whiskbroom imagers, frame imagers, and calibrated imagery.

In encoder, a compressed image consists of a header that encodes image and compression parameters followed by a body, that produced by an entropy coder which losslessly encodes the mapped prediction residuals. Entropy coder parameter are adaptively adjusted during this process to adapt changes of the mapped prediction residuals. Encoder encodes the mapped prediction residual by Sample or Block based adaptive entropy coding. The block-adaptive encoding approach was included as an option in the standard so that implementers could take advantage of existing space-qualified hardware implementations of this encoder.

In CCSDS-MHC, input hyperspectral image is chosen at the first place with a number of datasets from AVIRIS and Hyperion images. Once the image has been chosen, the code will compress the image through the encoded binary image form. The CCSDS-MHC algorithm focus only on the performance of an OpenMP parallelization in compression part. While the decompression part is taken to check the losslessness in hyperspectral image. However, Raspberry Pi can only perform compression operation due to memory performance whereby decompression part is inapt but can be performed with other multicore system. Overall data flow of Hyperspectral Image Compression is shown in Figure 3.
2.3 Coding Structure

In Listing 1, there is eight important function in developing an OpenMP code:

```c
1. #pragma omp parallel for
2. prepareNPreviousBands
3. ec_init
4. prepareBands
5. predictor_compress
6. codeSample
7. ecupdate
8. numBitsWritten

```#pragma omp parallel for schedule(static,band/omp_get_max_threads())
reduction(+:numBitsWritten)
for(z = 0; z < band; z++)
prepareNPreviousBands(z, omp_get_max_threads(), ...);
ec_init(z, ...);
prepareBands(z, ...);
for(y = 0; y < row; y++)
for(x = 0; x < col; x++)
outputMappedResidual[x][y][z] = predictor_compress(x, y, z, ...);
numBitsWritten+=codeSample(outputMappedResidual[x][y][z], x, y, z, ...);
ecupdate(outputMappedResidual[x][y][z], x, y, z, ...);
}
}}
totalNumBitsWritten += numBitsWritten;
```

#pragma omp for only delegates portions of the loop for different threads in the current team. A team is the group of threads executing the program. At program start, the team consists only of a single member. To create a new team of threads, the parallel keyword must be specified. Therefore, syntax of #pragma omp parallel for in line 1 is used as the compiler directive to connect the communication to the compiler, making the next for loop (line 2) a parallel region.

For schedule clause, the directive parallel for schedule is used to assign contiguous range of interaction that are normally called chunks in each processor. There are four type of scheduling which is in dynamic, guided, runtime and static. In this project, schedule clause for static is used, specifying the size of the chunk in the processor. Therefore, the syntax of #pragma omp parallel for schedule (static,band/omp_get_max_threads) is declared.

prepareBands is the function that prepares the prediction neighbourhood by bands shifting. Before commencing the execution, the number of bands to perform in this operation is already being specified in the declaration section. The execution time for prepareNPreviousBands is too long compared to prepareBands. Therefore, in this project, schedule clause is performed to reduce the time execution that can combine both functions together at one time.

ec_init is the implementation that supports the accumulator initialize constant. In this project, the accumulator constant is specified in the declaration sections. predictor_compress is declared in this function to compress the data read from the prepareBands.
function consists of mathematical calculation to compress the data so that the execution time will be lower.

On CCSDS-MHC, codeSample is declared from the number of prediction bands where the data was taken from the compression bands. Data samples have a fixed-size of dynamic range of D bits where D should be the integer in the range 2 < D < 16 [14]. If the image data is read and the dynamic range is too small according to the range, error sign will appear in the program terminal.

Along the process to compress the bands, ecupdate is important to read back the fixed size of original sample data. ecupdate is in charge of rescaling the counter size of compression bands. Lastly, numBitsWritten is the data update for the image of compressed size after performing some mathematical calculation on a dynamic range value. All the important functions in developing an OpenMP code are as shown in Listing 1.

2.4 Execution Time
In this project, the execution time is used to measure and analyse the performance of CCSDS-MHC. The execution time consist of CPU time and real-world time. The I/O execution time which is known as CPU time is ignored because the focus on this research is only on the time for reading/writing the hyperspectral/compressed image from/to the memory which is known as real-world time.

For the execution time performance parameter, it is simply evaluated in seconds. When the computer is multitasking, the real time for each program is determined separately, and depends on how the microprocessor allocates resources among the programs. Therefore, there is a difference between real-world time and CPU time in an embedded system. To get the real time to be used by a task within a C application, omp_get_wtime function is needed to measure a multi-threaded program as shown in Figure 4.

The omp_get_wtime function returns a double precision value equal to the number of seconds since the initial value of the operating system is the real-time clock. The initial value is guaranteed unchanging during the execution of the program. The value returned by the omp_get_wtime function is not consistent across all threads in the team [15].

2.5 Compression Ratio
Compression ratio is a term used to evaluate the reduction in data-representation size produced by CCSDS-MHC algorithm. Compression ratio is calculated as shown in Table 1 and is defined in equation (1).

\[
\text{Compression Ratio, CR} = \frac{\text{Size of Original Image}}{\text{Size of Compressed Image}}
\] (1)

The compression ratio in Table 1 is used as guide to verify the correct data of parallelization in CCSDS-MHC for hyperspectral image compression. The lossless compression performance through the bit-rate of the compressed image, in the number of bpppb. Note that both the original AVIRIS and Hyperion datasets have bit-rates of 16 bpppb [16]. The bit-rate (in bpp) of the compressed data are defined in equation (2).

\[
bpp = \frac{\text{Original image bit-rate}}{\text{Compression Ratio}}
\] (2)

| Table 1. Compression ratio for different hyperspectral images using CCSDS-MHC |
3. Result & Discussion

In this project, the parallelized CCSDS-MHC algorithm using OpenMP and without OpenMP that implement inside Raspberry Pi Model 3 B+. Section 3.1 is the result measured and verification of the lossless in terms of execution time. Raspberry Pi only perform the operation through the compression part meanwhile, the decompression part is not the interest in this project. Section 3.2 shows the result of the execution time when running in C/C++ language with OpenMP algorithm whereby different kinds of desktop consist of multicore system and data is executed based on the performances.

3.1 CCSDS-MHC using OpenMP

High data of hyperspectral images with a multithreaded function can increase the execution time for the system to read and write, especially upon performances in a low-power computer. To optimize the execution time for the system to operate in a low-power computer with high data, OpenMP in CCSDS-MHC algorithm is implemented.

Therefore, Table 2 show the execution time when running the CCSDS-MHC algorithm with and without OpenMP with hyperspectral image. Table 2 proves that there is a higher difference of execution time when OpenMP is applied in CCSDS-MHC algorithm. This is because OpenMP divides a task among the thread so that each thread executes its own allocated part of the code. This OpenMP is achieved by the parallelism task and data on CCSDS-MHC algorithm.

The speed performance and efficiency of CCSDS-MHC algorithm using the parallelization of OpenMP through its implementation inside Raspberry Pi 3 Model B+ is reported as average and percentage values. Running the CCSDS-MHC algorithm in Raspberry Pi with 4-threaded CPU-core without OpenMP for AVIRIS 1997 dataset was able to occupy the average of 289.0340 seconds with 88.4737 seconds using the OpenMP. The execution time for AVIRIS 1997 dataset increased to about 69.4% optimization when OpenMP is implemented.

For AVIRIS 2006 dataset, the average reading for CCSDS-MHC algorithm without OpenMP is 289.0340 seconds and 88.8774 seconds is achieved with OpenMP. It shows that AVIRIS 2006 dataset execution time is optimized to 69.3% when using the OpenMP.
For Hyperion image dataset, the average of 59.6981 seconds is recorded when the CCSDS-MHC algorithm is not implemented using OpenMP meanwhile, 19.2643 seconds is obtained when it is implemented with the OpenMP, optimizing the execution time to about 67.7%.

To summarize, it is proven that OpenMP can be exploited in CCSDS-MHC algorithm and compute faster results to about half of the execution time efficiency in Raspberry Pi 3 Model B+ with a low memory embedded system.

### Table 2. Execution time with and without OpenMP in Raspberry Pi 3

| Hyperspectral Image | CCSDS-MHC algorithm without OpenMP (seconds) | CCSDS-MHC algorithm with OpenMP (seconds) | Improvement (%) |
|---------------------|----------------------------------------------|------------------------------------------|-----------------|
| **AVIRIS 1997 dataset** | | | |
| Cuprite1            | 287.8375                                     | 88.3160                                   | 69.3            |
| Jasper1             | 285.5075                                     | 88.0509                                   | 69.2            |
| Low1                | 290.5157                                     | 88.8951                                   | 69.4            |
| Low5                | 290.2677                                     | 88.5578                                   | 69.5            |
| Lunar1              | 290.0447                                     | 88.5488                                   | 69.4            |
| **Average**         | **288.8346**                                 | **88.4737**                               | **69.4**        |
| **AVIRIS 2006 dataset** | | | |
| YSCal0              | 288.8649                                     | 89.1358                                   | 69.1            |
| YSCal3              | 290.2122                                     | 89.3906                                   | 69.2            |
| YSCal10             | 288.2797                                     | 88.8543                                   | 69.2            |
| YSCal11             | 289.3909                                     | 88.0401                                   | 69.6            |
| YSCal18             | 288.4221                                     | 88.9664                                   | 69.2            |
| **Average**         | **289.0340**                                 | **88.8774**                               | **69.3**        |
| **Hyperion dataset** | | | |
| Atturbah            | 60.5935                                      | 18.8207                                   | 68.9            |
| Benoni              | 60.4675                                      | 19.5144                                   | 67.7            |
| Boston              | 58.9004                                      | 19.0454                                   | 67.7            |
| Coolamon            | 59.4605                                      | 19.1911                                   | 67.7            |
| Dubbo1              | 60.1562                                      | 18.9905                                   | 68.4            |
| Edenton             | 60.7881                                      | 19.4695                                   | 68.0            |
| Greenland           | 58.2046                                      | 19.2856                                   | 66.9            |
| Maizhokunggar       | 59.0472                                      | 19.6949                                   | 66.7            |
| Okha                | 59.9177                                      | 19.6098                                   | 67.3            |
| Portobago           | 59.4457                                      | 19.0207                                   | 68.0            |
| **Average**         | **59.6981**                                  | **19.2643**                               | **67.7**        |

### 3.2 Performances with Other Implementations

This project consists of four types of different platform used which are Raspberry Pi 3 Model B+, Desktop with 4 threads, Desktop with 8 threads and Desktop with 16 threads. For specifications, Raspberry Pi 3 Model B+ processor consists of Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit @ 1.4GHz quad-core processor, Intel® Core™ i5-3210M CPU @ 2.50GHz, 2-core processor for the Desktop with 4 threads, Intel® Core™ i7-2600S for the Desktop with 8 threads and Intel® Xeon® CPU E5-1660 v4 @ 3.20GHz for the Desktop with 16 threads.

OpenMP compiler directives and without any major source code modifications can easily be ported to run in parallel on any shared-memory, multicore systems with reasonably good computational performance. The shared-memory of the multicore architecture is the state-of-the-art hardware
environment for modern microprocessors, including computational nodes of workstation, personal desktop and laptop [17].

The execution time performances and efficiency of lossless hyperspectral image compression that implements the OpenMP in CCSDS-MHC algorithm with low memory computerization using Raspberry Pi 3 Model B+ and high memory computerization through various threads of desktop are shown in Table 3.

Performance and efficiency of OpenMP parallel implementation depend on the hardware system as well whereby all of the maximum thread in the system is used. Time execution is measured by implementing CCSDS-MHC algorithm on different processor performance. The performance and efficiency of CCSDS-MHC algorithm using the parallelization of OpenMP that is implemented inside Raspberry Pi 3 Model B+ and with desktop of 16 threads, 8 threads as well as 4 threads of time execution is measured and reported in terms of average and percentage. The result of the performances (in percentage) is obtained from the value of the average execution time for each Desktop which is made in comparison with the Raspberry Pi 3 Model B+ data. It is noted that, all the processor threads are used to run the CCSDS-MHC algorithm for lossless hyperspectral image compression.

When running the CCSDS-MHC algorithm for AVIRIS 1997 image dataset, the Raspberry Pi 3 B+ managed to be executed with the average reading of 88.4737 seconds whereas 18.0756 seconds for the Desktop with 4 threads, 10.7640 seconds for the Desktop with 8 threads and 4.7184 seconds for the Desktop with 16 threads. By running the CCSDS-MHC algorithm on a Desktop of 4 threads with 2 CPU-core, the mean parallel efficiency is 79.6%. In contrast to 4-thread, the 8-thread with 4 CPU-core has the mean parallel efficiency of 87.8%. When 16 threads with 8 CPU-core are used, the parallelization of CCSDS-MHC algorithm has improved by 94.7% as compared to the 4 CPU cores of Raspberry Pi 3 B+.

For AVIRIS 2006 image dataset, the average that was obtained from Raspberry Pi 3 is 88.8774 seconds, 17.9008 seconds for the 4-thread Desktop, 10.6580 seconds for the 8-thread Desktop and 4.7870 seconds for the Desktop with 16 threads. By running the CCSDS-MHC algorithm on a Desktop of 4 threads with 2 CPU-core, the mean parallel efficiency is 79.9%. In contrast to the 4 threads, the 8 threads with 4 CPU-core has the mean parallel efficiency of 88.0%. When 16-thread Desktop with 8 CPU-core are used, the parallelization of CCSDS-MHC algorithm has accelerated by 94.6% compared to the 4 CPU cores of Raspberry Pi 3 B+.

However, using Hyperion image dataset that were cropped to a size of 256 × 256 pixels, only 196 out of 242 bands were classified on the execution time for Hyperion images whereby it is lower compared to AVIRIS images. Therefore, the average of the execution time obtained for Raspberry Pi 3 B+ is 19.2643 seconds, 3.7439 seconds for the Desktop with 4 threads, 1.5257 seconds for the Desktop 8 threads and 0.6143 seconds for the Desktop with 16 threads. Comparing the mean performances with Raspberry Pi 3 B+ execution time, the 4-thread with 2 CPU-cores obtained 80.6% and 92.1% for the 8-thread with 6 CPU-cores. When the 16-thread with 8 CPU-cores are used, the parallelization of CCSDS-MHC algorithm has accelerated by 96.8% compared to the 4 CPU cores of Raspberry Pi 3 B+.

The optimizing performance for the 16-thread Desktop with 8 CPU-cores is increased to 96.8% which is the percentage of the performance that furnishes a large difference compared to AVIRIS 1997 dataset which has a difference of 2.1% and a difference of 2.2% for AVIRIS 2006 dataset.

The differences in performances of the execution time for every different embedded platform are due to the CPU type, CPU-core processor, difference in CPU thread and number of RAM. It can be concluded that, the number of processor threads available gives a large effect on the execution time of CCSDS-MHC algorithm in performing the task.
Table 3. Time execution with other implementation

| Hyperspectral Image  | Desktop (16 threads)a | Desktop (8 threads)b | Desktop (4 threads)c | Raspberry Pi 3 B+ (4 threads)d |
|----------------------|-----------------------|----------------------|----------------------|--------------------------------|
| AVIRIS 1997 dataset  |                       |                      |                      |                                |
| Cuprite1             | 4.5000s               | 10.8100s             | 17.8910s             | 88.3160s                       |
| Jasper1              | 4.6400s               | 10.6390s             | 18.2290s             | 88.0509s                       |
| Low1                 | 4.9060s               | 10.8110s             | 18.1640s             | 88.8951s                       |
| Low5                 | 4.8740s               | 10.8420s             | 18.2350s             | 88.5578s                       |
| Lunar1               | 4.6720s               | 10.7180s             | 17.8590s             | 88.5488s                       |
| Average              | 4.7184s               | 10.7640s             | 18.0756s             | 88.4737s                       |
| Comparison*          | 94.7%                 | 87.8%                | 79.6%                |                                |
| AVIRIS 2006 dataset  |                       |                      |                      |                                |
| YSCal0               | 4.7180s               | 10.7170s             | 17.9640s             | 89.1358s                       |
| YSCal3               | 4.7960s               | 10.5930s             | 17.9660s             | 89.3906s                       |
| YSCal10              | 4.8740s               | 10.6230s             | 17.8330s             | 88.8543s                       |
| YSCal11              | 4.7810s               | 10.6550s             | 17.8540s             | 88.0401s                       |
| YSCal18              | 4.7660s               | 10.7020s             | 17.8870s             | 88.9664s                       |
| Average              | 4.7870s               | 10.6580s             | 17.9008s             | 88.8774s                       |
| Comparison*          | 94.6%                 | 88.0%                | 79.9%                |                                |
| Hyperion dataset     |                       |                      |                      |                                |
| Atturbah             | 0.6100s               | 1.5290s              | 3.7400s              | 18.8207s                       |
| Benoni               | 0.6250s               | 1.5130s              | 3.7460s              | 19.5144s                       |
| Boston               | 0.6250s               | 1.6070s              | 3.7520s              | 19.0454s                       |
| Coolamon             | 0.6250s               | 1.5130s              | 3.7970s              | 19.1911s                       |
| Dubbo1               | 0.6100s               | 1.5440s              | 3.7210s              | 18.9905s                       |
| Edenton              | 0.6100s               | 1.4980s              | 3.7090s              | 19.4695s                       |
| Greenland            | 0.6100s               | 1.5130s              | 3.7190s              | 19.2856s                       |
| Maizhokunggar        | 0.6090s               | 1.4970s              | 3.7190s              | 19.6949s                       |
| Okha                 | 0.6090s               | 1.4980s              | 3.7630s              | 19.6098s                       |
| Portobago            | 0.6100s               | 1.5450s              | 3.7730s              | 19.0207s                       |
| Average              | 0.6143s               | 1.5257s              | 3.7439s              | 19.2643s                       |
| Comparison*          | 96.8%                 | 92.1%                | 80.6%                |                                |

*Comparison between a, b, c with respect to d

4. Conclusion

With the support of OpenMP in CCSDS-MHC algorithm, each thread was executed independently by the parallel section of the code which was found to be more efficient and allowed higher level of constructs into a simplified form. Table 2 proved that the parallelization of CCSDS-MHC algorithm using OpenMP was optimized to approximately half of the execution time compared to the CCSDS-MHC algorithm without OpenMP. This proved that in this research, CCSDS-MHC algorithm by using OpenMP was successfully developed and can be implemented inside the Raspberry Pi 3 Model B+.

The execution time performances with others implementation results in Table 3 shown that performing CCSDS-MHC algorithm with OpenMP was able to obtain the averages of 88 seconds for AVIRIS and 19 seconds for Hyperion images. Besides, when implemented using Desktop with different types of multicore, the execution time for hyperspectral image compression decreases depending on the Desktop specification itself such as processor speeds and architectures as well as RAM sizes. This proved that the validation and measurement of the performance for losslessness hyperspectral image compression algorithm using CCSDS-MHC was successful and the objectives of this study are achieved.

Acknowledgement

This work was supported by Universiti Sains Malaysia Short Term Grant 304/PELECT/6315003.
Reference

[1] D. A. Landgrebe, *Signal Theory Methods in Multispectral Remote Sensing*. New Jersey: John Wiley & Sons, 2003.

[2] G. McNamara, J. M. Larson, S. A. Schwartz, and M. W. Davidson. (2019, 14th. Oct.). *Spectral Imaging and Linear Unmixing*. Available: https://www.microscopyu.com/techniques/confocal/spectral-imaging-and-linear-unmixing

[3] M. I. Afjal, M. A. Mamun, and M. P. Uddin, "Band reordering heuristics for lossless satellite image compression with 3D-CALIC and CCSDS," *Journal of Visual Communication and Image Representation*, vol. 59, pp. 514-526, 2019/02/01/ 2019.

[4] G. Lopez, B. Napoli, and A. G. M. Strollo, "FPGA implementation of the CCSDS-123.0-B-1 lossless Hyperspectral Image compression algorithm prediction stage," in 2015 IEEE 6th Latin American Symposium on Circuits & Systems (LASCAS), 2015, pp. 1-4.

[5] J. E. Sanchez, E. Auge, J. Santalo, I. Blanes, J. Serra-Sagrista, and A. Kiely, "Review and Implementation of the Emerging CCSDS Recommended Standard for Multispectral and Hyperspectral Lossless Image Coding," presented at the Proceedings of the 2011 First International Conference on Data Compression, Communications and Processing, 2011.

[6] M. Olaru and M. Craus, "Lossless multispectral and hyperspectral image compression on multicore systems," in 2017 21st International Conference on System Theory, Control and Computing (ICSTCC), 2017, pp. 175-179.

[7] H. Du, W. Zhang, N. Guan, and W. Yi, "Scope-aware data cache analysis for OpenMP programs on multi-core processors," *Journal of Systems Architecture*, vol. 98, pp. 443-452, 2019/09/01/ 2019.

[8] S. Balakrishnan, D. Langelman, E. Goretok, and D. A. D. George, "Deep Learning for Hyperspectral Image Classification on Embedded Platforms," in 2018 IEEE International Conference on Image Processing, Applications and Systems (IPAS), 2018, pp. 187-191.

[9] N. Kefalas and G. Theodoridis, "Low-memory and high-performance architectures for the CCSDS 122.0-B-1 compression standard," *Integration*, 2018/04/10/ 2018.

[10] R. L. Davidson and C. P. Bridges, "GPU accelerated multispectral EO imagery optimised CCSDS-123 lossless compression implementation," in 2017 IEEE Aerospace Conference, 2017, pp. 1-12.

[11] B. Hopson, K. Benkrid, D. Keymeulen, and N. Aranki, "Real-time CCSDS Lossless Adaptive Hyperspectral Image Compression on Parallel GPGPU & Multicore Processor Systems," in NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2012, Erlangen, Germany, 2012, pp. 107-114.

[12] S. Wang, S. Zhang, X. Huang, J. An, and L. Chang, "A Highly Efficient Heterogeneous Processor for SAR Imaging," *Sensors*, vol. 19, no. 15, p. 3409, 2019.

[13] N. Lopez-Ruiz, F. Granados-Ortega, A. Carvajal Miguel, and A. Martinez-Olmos, "Portable multispectral imaging system based on Raspberry Pi," *Sensor Review*, vol. 37, no. 3, pp. 322-329, 2017.

[14] CCSDS, *Lossless Multispectral & Hyperspectral Image Compression - Recommended Standard (CCSDS 123.0-B-1) (no. 1)*. Washington DC, USA: CCSDS Secretariat, NASA, 2012, p. 52.

[15] SaiSang et al. (2019, 14th. Oct.). *Run-time library functions*. Available: https://docs.microsoft.com/en-us/cpp/parallel/openmp/3-run-time-library-functions?view=vs-2019

[16] N. R. Mat Noor, "Karhunen-Loève Transform based Lossless Hyperspectral Image Compression for Space Applications," PhD eThesis, Department of Engineering, University of Leicester, Leicester, 2015.

[17] X. Xu, "OpenMP parallel implementation of stiffly stable time-stepping projection/GMRES(ILU(0)) implicit simulation of incompressible fluid flows on shared-memory, multicore architecture," *Applied Mathematics and Computation*, vol. 355, pp. 238-252, 2019/08/15/ 2019.