GPU Acceleration of Predictive Partitioned Vector Quantization for Ultraspectral Sounder Data Compression

Shih-Chieh Wei and Bormin Huang

***Abstract*—For the large-volume ultraspectral sounder data, compression is desirable to save storage space and transmission time. To retrieve the geophysical paramters without losing precision the ultraspectral sounder data compression has to be lossless. Recently there is a boom on the use of graphic processor units (GPU) for speedup of scientific computations. By identifying the time dominant portions of the code that can be executed in parallel, significant speedup can be achieved by using GPU. Predictive partitioned vector quantization (PPVQ) has been proven to be an effective lossless compression scheme for ultraspectral sounder data. It consists of linear prediction, bit depth partitioning, vector quantization, and entropy coding. Two most time consuming stages of linear prediction and vector quantization are chosen for GPU-based implementation. By exploiting the data parallel characteristics of these two stages, a spatial division design shows a speedup of 72x in our four-GPU-based implementation of the PPVQ compression scheme.**

***Index Terms*—Graphic processor unit, lossless data compression, predictive partitioned vector quantization, ultraspectral sounder data.**

I. Introduction

CONTEMPORARY ultraspectral sounders such as the Atmospheric Infrared Sounder (AIRS) [1], the Infrared Atmospheric Sounding Interferometer (IASI) [2], the Cross-Track Infrared Sounder (CrIS) [3], and the Geosynchronous Imaging Fourier Transform Spectrometer (GIFTS) [4] are designed in an aim to improve the weather and climate prediction. These ultraspectral sounders feature a spectral resolution of over a thousand infrared channels in each spatial location. Thus the large volume of the 3D data observed each day will need some form of compression to reduce its size for data transfer and archive. On the other hand, the physical retrieval of the geophysical parameters from the observation data involves the inverse solution of the radiative transfer equation, which is a mathematically ill-posed problem and the solution is sensitive to the error or noise in the data [5]. Therefore, there is a need for the compression to be lossless to avoid potential retrieval degradation of geophysical parameters due to lossy compression.

Recently general purpose graphic processor units (GPGPU) have become more affordable for scientific computation. In general, a GPU board may contain many multiprocessors which are composed of several execution cores, registers and shared memory [6], [7]. Each multiprocessor can execute the same code on its cores independently. By exploiting the fast registers and shared memory in the multiprocessor, parallel codes can usually have high speedup in computation. GPUs have already seen many applications in traditional visual rendering [8]. There are also trends of using GPU in other compute intensive applications like image compression [9], [10]. In remote sensing, GPUs have been used to accelerate the hyperspectral image processing [11] which includes compression [12], spectral mixture analysis [13], and target and anomaly detection [14].

The predictive partitioned vector quantization (PPVQ) [15] is known for its high performance on lossless compression of ultraspectral sounder data. It mainly consists of the four stages of linear prediction, bit depth partitioning, vector quantization and entropy coding. Based on a C code implementation of PPVQ, we analyze the CPU profile of the four stages on running the GIFTS data [16] and find out that the two stages of linear prediction and vector quantization take up over 90 percent of the CPU execution time. In this paper, we will seek to exploit the parallelism available in these two stages for speedup of PPVQ. In particular we will use the CUDA platform on the Nvidia graphics hardware for implementation.

The rest of this paper will be arranged as follows. Section II describes the GIFTS test data set used in this study. Section III introduces the PPVQ compression scheme and our GPU-based implementation of it. Section IV shows the experimental results and Section V gives conclusions.

II. Data

GIFTS represents a revolutionary step in satellite remote sensing of geophysical parameters, and poses a challenge to process the large amount of data it will collect. Most of a raw GIFTS data "cube" is made up of a 128×128 array of interferograms. Metadata and a visible light image also included in each cube take up a negligible portion of its volume. Each of the 16384 detector pixels contributes two separate complex-valued interferograms, each from one of the instrument's two infrared detector arrays: The long wave (LW) interferogram records 1031 data values per pixel, while the short-medium wave (SMW) interferogram records 2062 values. The interferograms

Manuscript received September 01, 2010; revised November 30, 2010; accepted January 07, 2011. Date of publication May 02, 2011; date of current version August 26, 2011.

S.-C. Wei was with the Space Science and Engineering Center, University of Wisconsin-Madison. He is currently with the Department of Information Management, Tamkang University, Tamsui 251, Taiwan.

B. Huang is with the Space Science and Engineering Center, the University of Wisconsin-Madison, Madison, WI 53706, USA (e-mail: bormin@ssec.wisc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSTARS.2011.2132117

TABLE I
FIVE OBSERVATION DATA CUBES USED IN THE STUDY

Data Cube	Name	Time (MST)
1	Ave00004	12:10
2	Ave00015	14:00
3	Ave00027	16:00
4	Ave00039	18:00
5	Ave00051	20:00

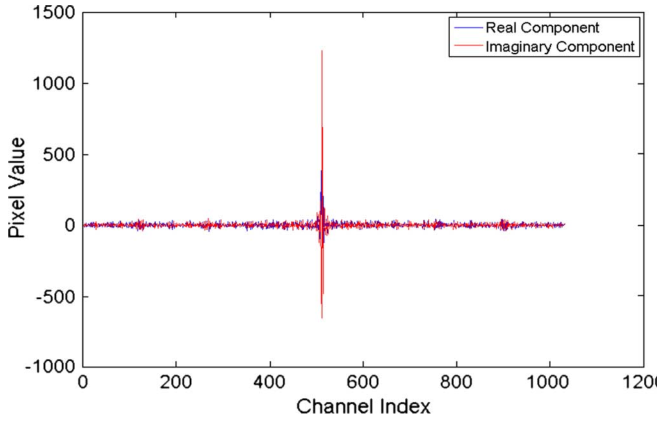


Fig. 1. Sample interferograms of representative pixels (located near center of LW array).

are complex valued as the result of a filtering algorithm implemented in an onboard DSP that downsamples raw counts from the interferometer to a format usable by downstream product generation algorithms. The real and imaginary parts of each value are stored separately as 16-bit (short) integers, resulting in 32 bits (4 bytes) per raw value, or 12 kilobytes per pixel. This raw format results in 192 MB of infrared interferometric data per observation, or "cube". In its normal operating mode, GIFTS is expected to generate one such cube every 11 seconds, and this data generation rate, on the order of 1.5 TB a day, demands the kind of sophisticated compression for transmission and storage that is presented in this paper.

The test dataset prepared for this compression study was gathered in 2006 in a ground testing facility, and consists of uplooking instrument views taken at various times over the course of a clear sky day. While spatial uniformity of the samples is fairly high due to the small angle of view of the instrument, the spectral dimension in each pixel records the same kind of complex radiative signature as seen from outer space. But due to the nature of radiative transfer, this spectrum is the inverse of its downlooking equivalent. All samples were taken on September 13, 2006 in the Space Dynamics Lab testing facility in Logan, Utah [17]. Table I lists the five observation data cubes used in the study. Fig. 1 shows the sample interferograms of representative pixels (located near center of LW array). Fig. 2 shows the spatial variability of raw interferogram sampled in off-peak region.

III. A GPU-BASED IMPLEMENTATION OF PPVQ FOR GIFTS DATA

A. PPVQ Profile

The predictive partitioned vector quantization (PPVQ) is known for its effective compression of ultraspectral sounder data [15], [16]. PPVQ consists of the following four major stages in performing the compression.

1) Linear prediction (LP). This stage can reduce the dynamic range of a pixel by knowledge of its previous channels [18].
2) Bit depth partitioning (BP). This stage groups the channel residual by bit depth. Channels with the same bit depth are assigned to the same partition and VQ is applied to each partition separately. Fig. 3 shows the bit depth of interferograms before and after the bit depth partitioning stage.
3) Vector quantization (VQ). This stage divides high-dimensional data (vectors) into groups having approximately the same number of points closest to them. Each group is represented by its centroid point [19], [20].
4) Entroy coding (AC). This stage assigns codes to symbols so as to match code lengths with the probabilities of the symbols [21], [22]. AC is applied on the VQ output of each bit depth partition which includes the codebook, residual, and the index.

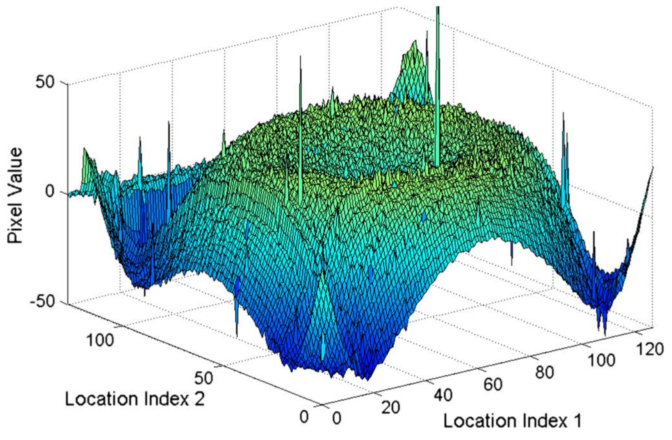


Fig. 2. Spatial variability of raw interferogram sampled in off-peak region.

For PPVQ, the profile of CPU time on GIFTS data is shown in Fig. 4. It can be seen that among the four stages, the two stages of linear prediction (LP) and vector quantization (VQ) take up most of the CPU execution time. I/O refers to the disk time spent on reading input and writing output which is almost negligible. Therefore in the following sections we will focus on the GPU implementation of the two stages.

B. Linear Prediction

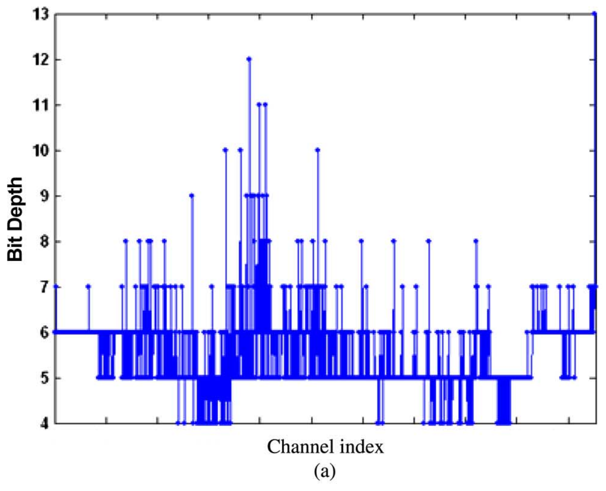
The spatial frame X of channel i can be linearly predicted by $\hat{X}_i$ in (1) as follows:

$$\hat{X}_i = \sum_{k=1}^{n_p} c_k X_{i-k} \quad \text{or} \quad \hat{X}_i = X_p C \tag{1}$$

where $\hat{X}_i$ is the vector of channel i representing the predicted 2D spatial frame, X_p is the matrix consisting of n_p neighboring channels, and C is the vector of the prediction coefficients for channel i. The prediction coefficients C can be obtained from (2) as follows: [23]

$$C = \left(X_p^T X_p\right)^{-1} \left(X_p^T \hat{X}_i\right). \tag{2}$$

The prediction error is the difference between the original channel vector and its predicted counterpart.



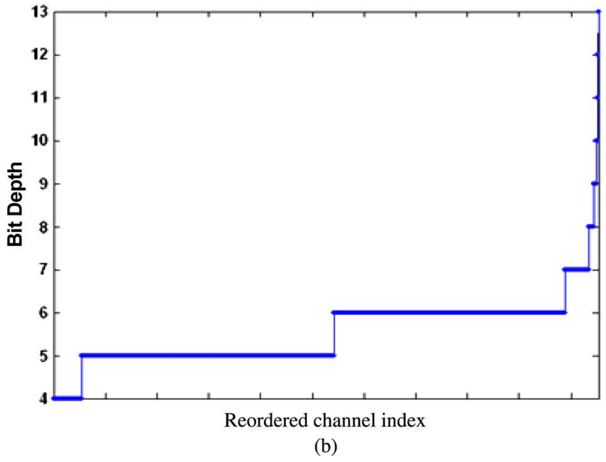


Fig. 3. The bit depth of interferograms before (a) and after (b) bit depth partitioning.

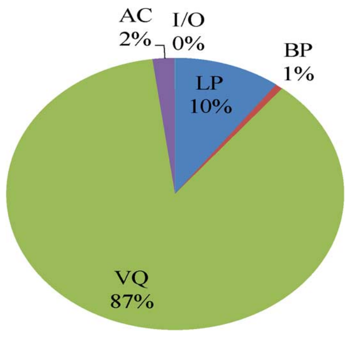


Fig. 4. The average CPU profile of running PPVQ on GIFTS data cubes.

For GPU implementation, three kernels are used for performing the linear prediction in the above formulation. The first kernel prepares the matrix $A = X_p^T X_p$ and the vector $B = X_p^T X_i$ where X_p consists of the previous neighboring n_p channel frames of channel i. The second kernel solves the linear equation $AC = B$ for the n_p prediction coefficients in solution vector C. The third kernel then computes the linear prediction residual $E_i = X_i - \hat{X}_i$ where $\hat{X}_i = X_p C$ is the linearly predicted frame of channel i.

Among the 3 kernels, computation of A and B is most time consuming. For speedup, multiple GPUs can be used to share the computation workload. Also, for $n_c = 1031$ and $n_p = 32$, frames of channels $0 \sim 31$ can be processed on CPU. A simple difference operation is performed on the first channel, and the linear prediction by available previous channels is performed on the remaining channels.

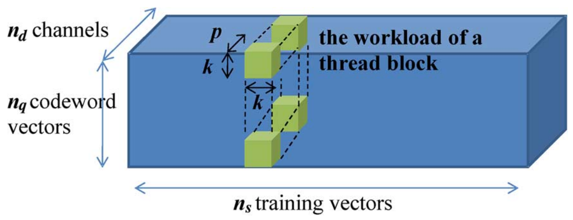


Fig. 5. The schematic of a thread block sharing the workload in codeword assignment. There are n_s training vectors and each needs to find the smallest one of n_q codewords in distortion. A thread block shares the workload of k training vectors using k by p threads. All vectors have a dimension of n_d channels.

C. Vector Quantization (VQ)

The vector quantization consists of codebook training and codeword encoding. The codebook training uses the well-known Linde–Buzo–Gray (LBG) algorithm [19] but for fast learning, just takes the first training vector in each group as the initial codeword of the codebook. LBG is an iterative process of codeword assignment and codeword computation. In particular, codeword assignment is most time-consuming for GIFTS. Given a codebook of n_q codeword vectors, the task of codebook assignment is to assign the smallest codeword in distortion to each of the n_s training vectors. All vectors have a dimension of n_d channels. For GIFTS data, typical values are $n_s = 16384$; $n_q = 256, 512, 1024$, or 2048; and $n_d = 1 \sim 500$.

Each multiprocessor of the GPU executes a thread block independently. Thus the workload of codeword assignment can be shared by thread blocks as shown in Fig. 5. A block of k by p threads computes the distortions between k training vectors and all n_q codeword vectors. The computation involves the sweeping of a tile of $k \times p$ threads throughout a $k \times k \times p$ cube, or the yellow region in Fig. 5. At each iteration, the tile first loads the partial p-channel components of k training vectors and k codeword vectors into the shared memory. Then it loops along the k training vectors to compute the difference squared and distortion between each training vector and the k codeword vectors. The tile then moves to the next p-channel components until all n_d-channel distortion are computed. At this point each of the k training vectors can mark the smallest codeword in distortion of the k codeword vectors seen previously. The tile then moves to the next k codewords for a similar comparison operation until all n_q codewords are seen and the smallest codeword in distortion is assigned.

In this way, the tile first moves along the n_s dimension in 1-training-vector units, then the n_d dimension in p-channel units, and finally the n_q dimension in k-codeword units. Shared variables in use include the $k \times p$ partial codewords, the $k \times p$ partial training vectors, the $k \times p$ partial difference vectors, and the $k \times k$ partial distortions. The size of the thread block is limited by the size of the shared memory and the maximum number of threads allowed by the GPU hardware.

As the vector quantization of each bit depth partition is independent, multiple GPUs can be exploited for speedup. Each GPU is assigned to an independent thread. An idle thread can retrieve a bit depth partition from a thread-safe bit depth partition

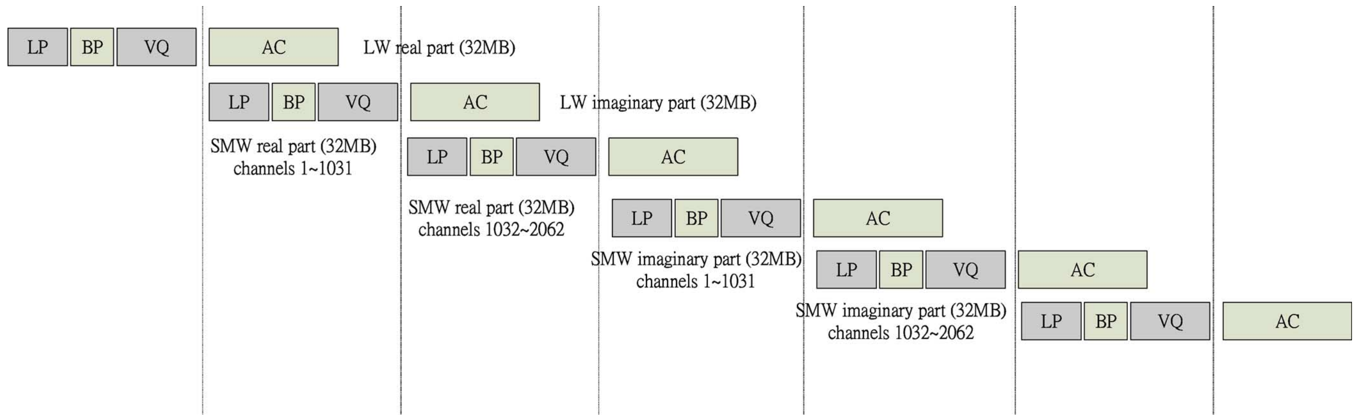

LP
BP
VQ
AC
LW real part (32MB)
LW imaginary part (32MB)
SMW real part (32MB) channels 1~1031
SMW real part (32MB) channels 1032~2062
SMW imaginary part (32MB) channels 1~1031
SMW imaginary part (32MB) channels 1032~2062

Fig. 6. The pipeline design of PPVQ for GIFTS data.

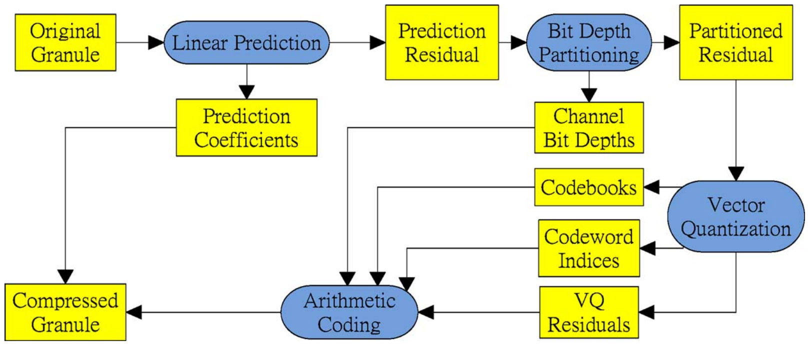


Fig. 7. The data flow diagram for the PPVQ compression scheme.

queue to do vector quantization on its own GPU. The VQ output of codebook, residual, and index is appended to a thread-safe AC queue for later compression on CPU.

D. Pipleline Design

In one observation, a GIFTS data cube of 192 MB is obtained which is divided into 6 subcubes of equal size 32 MB, each containing 1031 channels. The 6 subcubes are the LW real part, the LW imaginary part, the SMW real part of channels $1 \sim 1031$, the SMW real part of channels $1032 \sim 2062$, the SMW imaginary part of channels $1 \sim 1031$, and the SMW imaginary part of channels $1032 \sim 2062$. As shown in Fig. 6, each subcube goes through the LP, BP, VQ and AC steps sequentially for compression. The LP and VQ steps are mainly done on GPU, while the BP and AC steps are done on CPU. The length of each step is not drawn to scale. The pipeline is synchronized (denoted by the 6 vertical dotted lines) such that when one subcube is doing AC, the next subcube is doing LP, BP, and VQ. Overlap of the LP, BP, VQ steps with the AC step in time can reduce the total processing time of a data cube.

E. Spectral vs. Spatial Division Design for Multiple GPUs

When there are more than one GPU available, more speedup can be achieved by using multiple GPUs simultaneously. For division of the compression workload among several GPUs, the two approaches of spectral division and the spectral division are proposed. Given n_g GPUs available, spectral division assigns each GPU to do the linear prediction for $(n_c - n_p)/n_g$ contiguous channels. In VQ stage, each GPU is responsible for quantization of a bit depth partition whose number of grouped channels might vary in size.

For spatial division design, each data cube is spatially divided into n_g subcubes. Then each subcube is assigned an independent GPU to do the subsequent compression work.

IV. Results

In the experiment, we test the GPU-based PPVQ lossless compression scheme on the five GIFTS data cubes which we introduced earlier in the data section. Fig. 7 shows the dataflow diagram for the PPVQ compression scheme.

For prediction of channel values, we use a linear prediction of 32 predictors. Compression of the first few channels of a cube which have less than 32 previous channels is performed on CPU. Compression of the remaining channels which have full 32 previous channels is performed on GPU. Both CPU and GPU can do the compression in parallel. As result of the prediction, the residual which is the difference between the actual value and the predicted value is fed into the partitioned vector quantization module.

As BP involves a time consuming matrix transpose preprocessing, the transpose operation is now combined into LP on GPU as a post-processing step. The remaining BP operation is still left on CPU. Current BP time is about 1/6 the original BP time and so is not further moved to run on GPU.

The bit depth partitioning is performed on CPU which is based on the bit depths of the channel residual. Channel residuals of equal bit depths are grouped together and applied a vector quantization with a codebook size of the same bit depth.

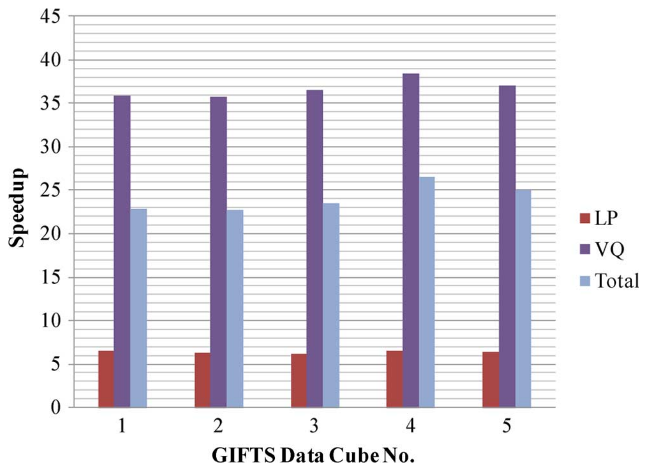


Fig. 8. The speedup profile of PPVQ using 1 GPU on 5 data cubes of the GIFTS test data.

For each partition a different codebook is trained on GPU by the fast VQ introduced earlier. The size of a thread block is chosen to be $k \times p = 51 \times 10 = 510$ threads. The choice of $k \times p = 51 \times 10$ instead of a better warp size conforming 64×8 is mainly due to the limited size of shared memory. A larger k will have a faster coverage of the required difference operations. But it also takes a larger buffer of k^2 numbers to store the distortion result. Therefore a 51×10 kernel is chosen under the constraint of 512 threads per block.

The secondary residual from the vector quantization is then compressed on CPU by the arithmetic coder. The experiment is carried out on a machine with a quad-core 2.4 GHz AMD CPU, and 4 Nvidia Tesla 1.3 GHz GPUs. Earlier we have developed a source code in C for PPVQ. For comparison, the single threaded C code was used as the CPU reference version. It was compiler-optimized by the gcc O3 optimization option. All GPU speedup was compared against this optimized CPU version. Using only one CPU core, the average compression time of the reference PPVQ version on a GIFTS data cube takes about 990 seconds.

Fig. 8 shows the speedup profile of PPVQ using 1 GPU on the 5 GIFTS data cubes. Using 1 GPU we have an LP speedup of 6×, a VQ speedup of 36×, and a total speedup of 24×. When there are more than one GPU available, two approaches of spectral division and spatial division design can be adopted.

Fig. 9 and Fig. 10 show the speedup profiles of PPVQ using 4 GPUs on the 5 GIFTS data cubes for the spectral division and the spatial division designs respectively. Using the pipeline design of LP and VQ on 4 GPUs and AC on CPU, the average compression time of the GPU-based PPVQ takes about 13 seconds for the spatial division design. Note that under the spatial division design, each subcube is smaller in size and thus the VQ and AC stages all involve less workload. Fig. 11 and Fig. 12 show the average speedup profiles over 5 data cubes from using 1 through 4 GPUs for the spectral division and the spatial division designs respectively.

A general trend can be seen that using more GPUs does give higher speedup for LP and VQ though each with different saturation curves. The fact that using 4 GPUs does not have a total speedup near 4 can be attributed to the reason that each GPU is not assigned a job of equal workload. For the spectral division

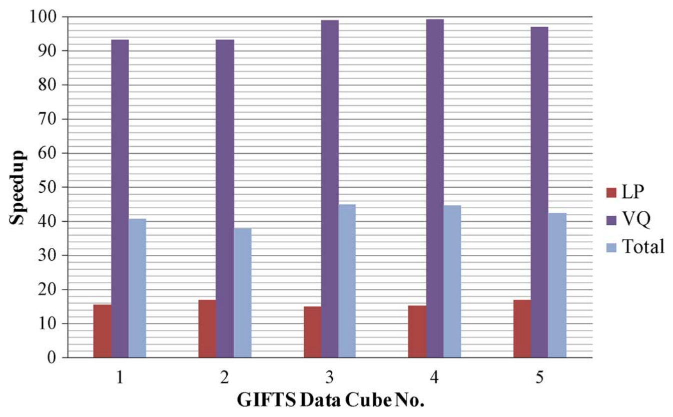


Fig. 9. For the spectral division design, the speedup profile of PPVQ using 4 GPUs on 5 data cubes of the GIFTS test data.

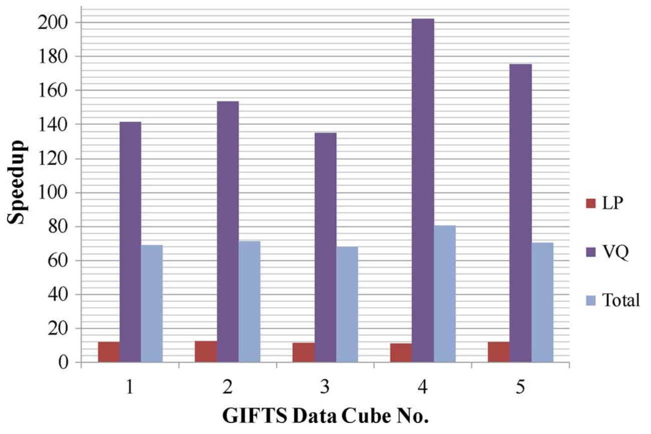


Fig. 10. For the spatial division design, the speedup profile of PPVQ using 4 GPUs on 5 data cubes of the GIFTS test data.

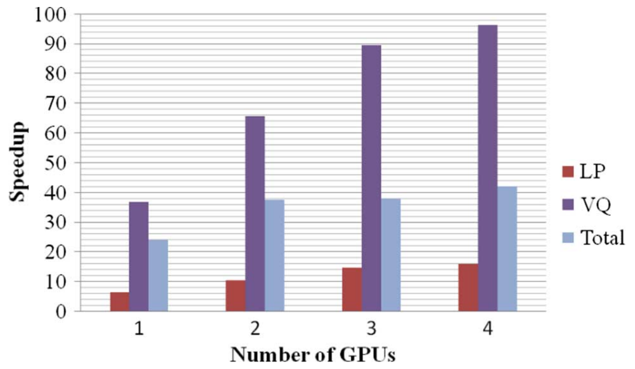


Fig. 11. The average speedup profile of PPVQ using spectral division by 1 through 4 GPU(s).

design, the VQ stage assigns one GPU to learning the codebook of a variable-sized bit-depth partition. As the size of the bit depth partition varies from one channel to 500 channels, the work load of each GPU will not be equal. For the spatial division design, one GPU is responsible for all VQ workload of a spatially divided subcube. As all subcubes are approximately equal in size and have about the same number of bit-depth partitions, each GPU tends to have more equal workload as shown in Fig. 12. Also in the pipeline design when LP and VQ on GPU already takes less time than AC on CPU, using more GPU will not solve the bottleneck on CPU. The latter case is reflected in the nearly

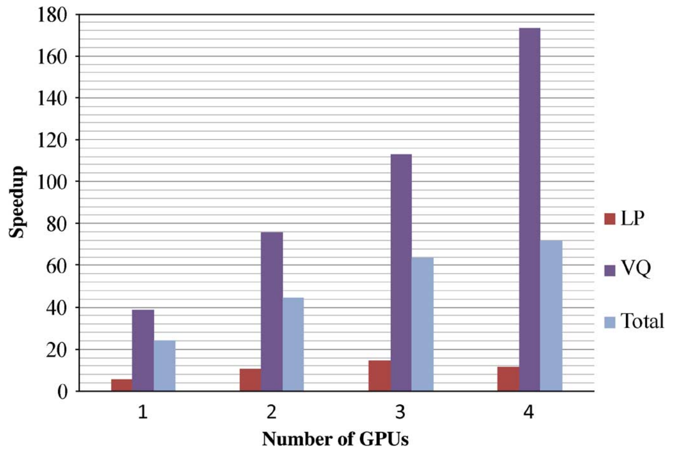


Fig. 12. The average speedup profile of PPVQ using spatial division by 1 through 4 GPU(s).

same total speedup using more than 2 GPUs in Fig. 11 and using more than 3 GPUs in Fig. 12.

V. Conclusions

The predictive partitioned vector quantization (PPVQ) compression scheme is known for its effectiveness in lossless compression of ultraspectral sounder data. Two most time consuming stages of PPVQ are identified based on an earlier developed C code. They are the linear prediction and vector quantization stages. In this paper, both stages are implemented on GPU. Furthermore an original GIFTS data cube is divided into 6 equal-sized parts to facilitate CPU and GPU processing in pipeline. When more than one GPU is available, two approaches of spectral division design and spatial division design are proposed. As result, we demonstrated that for the spatial division design, compression of an observation GIFTS data cube can be finished on 4 GPUs in about 13 seconds or a speedup of 72×. Therefore the GPU-based implementation of PPVQ provides a low-cost, effective and efficient compression solution for GIFTS compression for rebroadcast use. The future work will consider the implementation of AC on GPU for further speedup.

References

[1] H. H. Aumann and L. Strow, "AIRS, the first hyper-spectral infrared sounder for operational weather forecasting," in *Proc. IEEE Aerospace Conf.*, 2001, vol. 4, pp. 1683–1692.

[2] T. Phulpin, F. Cayla, G. Chalon, D. Diebel, and D. Schlussel, "IASI onboard Metop: Project status and scientific preparation," in *Proc. 12th Int. TOVS Study Conf.*, Lorne, Victoria, Australia, 2002, pp. 234–243.

[3] H. J. Bloom, "The Cross-track Infrared Sounder (CrIS): A sensor for operational meteorological remote sensing," in *Proc. 2001 Int. Geoscience and Remote Sensing Symp.*, 2001, pp. 1341–1343.

[4] W. L. Smith, F. W. Harrison, D. E. Hinton, H. E. Revercomb, G. E. Bingham, R. Petersen, and J. C. Dodge, "GIFTS—The precursor geostationary satellite component of the future earth observing system," in *Proc. IGARSS'02*, 2002, vol. 1, pp. 357–361.

[5] B. Huang, W. L. Smith, H.-L. Huang, and H. M. Woolf, "Comparison of linear forms of the radiative transfer equation with analytic Jacobians," *Appl. Opt.*, vol. 41, no. 21, pp. 4209–4219, 2002.

[6] *"Nvidia Cuda Programming Guide,"* ver. 3.0, 2010.

[7] *"Nvidia Cuda Reference Manual,"* ver. 3.0, 2010.

[8] D. B. Kirk and W.-m. W. Hwu, *Programming Massively Parallel Processors: A Hands-On Approach*. San Diego, CA: Morgan Kaufmann, 2010.

[9] V. Simek and R. R. Asn, "GPU acceleration of 2D-DWT image compression in Matlab with CUDA," in *Proc. UKSIM European Symp. Computer Modeling and Simulation*, 2008, pp. 274–277.

[10] W. van der Laan, A. Jalba, and J. Roerdink, "Accelerating wavelet lifting on graphics hardware using CUDA," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 1, pp. 132–146, 2011.

[11] J. Setoain, M. Prieto, C. Tenllado, and F. Tirado, "GPU for parallel on-board hyperspectral image processing," *Int. J. High Perform. Comput. Applicat.*, vol. 22, no. 4, pp. 424–437, 2008.

[12] J. Mielikainen, R. Honkanen, P. Toivanen, and B. Huang, "GPUs for data parallel spectral image compression," in *Proc. SPIE Int. Soc. Opt. Eng.*, 2009, vol. 7455, p. 74550C.

[13] A. Plaza, J. Plaza, S. Sanchez, and A. Paz, "Lossy hyperspectral image compression tuned for spectral mixture analysis applications on NVidia graphics processing units," in *Proc. SPIE Int. Soc. Opt. Eng.*, 2009, vol. 7455, p. 74550F.

[14] A. Paz and A. Plaza, "Clusters versus GPUs for parallel target and anomaly detection in hypersepctral images," *EURASIP J. Adv. Signal Process.*, vol. 2010, 2010, 18 pages.

[15] B. Huang, A. Ahuja, and H.-L. Huang, "Predictive partitioned vector quantization for hyperspectral sounder data compression," in *Proc. SPIE Int. Soc. Opt. Eng.*, 2004, vol. 5548, pp. 70–77.

[16] B. Huang, S.-C. Wei, A. H.-L. Huang, M. Smuga-Otto, R. Knuteson, H. E. Revercomb, and W. L. Smith, Sr., "Lossless compression of the geostationary imaging Fourier transform spectrometer (GIFTS) data via predictive partitioned vector quantization," in *Proc. SPIE Int. Soc. Opt. Eng.*, 2007, vol. 6683, p. 66830E.

[17] H. E. Revercomb, W. L. Smith, D. C. Tobin, R. O. Knuteson, F. Best, J. K. Taylor, D. D. Turner, D. K. Zhou, R. A. Reisse, G. W. Cantwell, and J. Tansock, "GIFTS radiance validation from ground-based sky-viewing comparisons to AERI," presented at the Fourier Transform Spectroscopy and Hyperspectral Imaging and Sounding of the Environment Conf., Washington, DC, 2007, FTuA5, 1-55752-828-4, The Optical Society of America.

[18] J. Mielikainen, P. Toivanen, and A. Kaarna, "Linear prediction in lossless compression of hyperspectral images," *Opt. Eng.*, vol. 42, no. 4, pp. 1013–1017, Apr. 2003.

[19] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantization design," *IEEE Trans. Commun.*, vol. COM-28, pp. 84–95, 1980.

[20] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Norwell, MA: Kluwer Academic, 1992.

[21] G. G. Langdon, "An introduction to arithmetic coding," *IBM. J. Res. Develop.*, vol. 28, pp. 135–139, Mar. 1984.

[22] I. H. Witten, R. M. Neal, and J. C. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, pp. 520–541, Jun. 1987.

[23] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD: The John Hopkins Univ. Press, 1996.

Shih-Chieh Wei received the B.E. degree in electrical engineering from National Taiwan University, Taiwan, in 1988, the M.S. degree in computer science from National TsingHua University, Taiwan, in 1990, and the Ph.D. degree in systems engineering from Osaka University, Japan, in 1998.

He was a visiting scholar at the Space Science and Engineering Center, the University of Wisconsin-Madison, and is currently an Assistant Professor in the Department of Information Management, Tamkang University, Taiwan.

His research interests include data compression, information retrieval, and GPU applications.

Bormin Huang received the M.S.E. degree in aerospace engineering from the University of Michigan, Ann Arbor, and the Ph.D. degree in the area of satellite remote sensing from the University of Wisconsin–Madison.

He was with the NASA Langley Research Center during 1998–2001, and is currently a research scientist and principal investigator at the Space Science and Engineering Center, University of Wisconsin–Madison. He has authored or coauthored over 100 scientific and technical publications. He has broad interest and experience in remote sensing science and technology.

Dr. Huang has been serving as a Chair for the SPIE Conference on Satellite Data Compression, Communications, and Processing since 2005, and a Chair for the SPIE Europe Conference on High Performance Computing in Remote Sensing since 2011. He currently serves as an Associate Editor for the *Journal of Applied Remote Sensing*. He has been a Session Chair or Program Committee member for several IEEE and SPIE conferences.