Feasibility Study of using BZIP2 within the FITS Tiled Image Compression Convention

W. D. Pence

NASA Goddard Space Flight Center, Greenbelt, MD 20771

William.Pence@nasa.gov

22 May 2009

ABSTRACT

The performance of the BZIP2 compression algorithm is studied to see if it is suitable for use within the FITS tiled-image compression convention, which already supports the Rice, Hcompress, and GZIP algorithms. Comparison tests of the compression ratios achieved by the different algorithms show that BZIP2 is relatively inefficient when the image is tiled on a row-by-row basis. BZIP2 typically requires larger blocks of data (at least several thousand image pixels) for it to achieve the same compression efficiency as the Rice algorithm. The timing comparison tests show that BZIP2 is an order of magnitude slower than the Rice algorithm. More specifically it requires 15 to 30 times more CPU time (depending on the amount of noise in the image) than Rice, or 4 to 6 times more than GZIP when compressing the image. When uncompressing the image, BZIP2 requires 4 to 6 times more CPU time than Rice. Based on these tests, there appears to be little justification for supporting the BZIP2 algorithm within the FITS tiled image compression convention.

1. Introduction

This research note studies the performance of the BZIP2 compression algorithm (Seward, J. 2007) to evaluate whether it should be supported within the FITS tiled-image compression convention (Pence et al. 2000; Seaman et al. 2007). The implementation of this convention within the CFITSIO library (Pence 1999) already supports 3 other general purpose compression algorithms – Rice, Hcompress, and GZIP – so the question is whether BZIP2 offers any unique advantages that would justify the additional cost to the FITS community of supporting it. These tests were performed using a customized version of the CFITSIO library which was modified to support version 1.0.5 of the BZIP2 code. The BZIP2 comparison tests reported here complement the work of Pence et al. (2009) which made a detailed comparison of the image compression performance of the Rice, Hcompress, and GZIP algorithms.

Under the FITS tiled-image convention, the image is first divided into a rectangular grid of "tiles". By default, the image is tiled on a row by row basis (except for Hcompress which

is inherently 2-dimensional in nature and uses 16 rows per tile); any other rectangular tile size may be specified, but the default size generally provides the best compromise between better compression (which favors large tiles) and faster random access to small sections of the image (which benefits from smaller tiles). Each tile of pixels is compressed using one of the available compression algorithms, and the compressed stream of bytes is stored in a variable length array column in a FITS binary table. Each row of the FITS binary table corresponds to one tile in the image.

2. Comparison of Compression Ratios

The compression ratio performance of the 4 different algorithms was measured on a sample of artificially constructed 16-bit integer images, each 500 by 500 pixels in size, where the pixels in each image have a constant value plus differing amounts of Gaussian-distributed random noise. The Gaussian σ of the noise in the images ranges from 1.0 to 500. As was shown in Pence et al. (2009), the effective number of noise bits per pixel in these images is given by

$$N_{bits} = \log_2(\sigma\sqrt{12}) = \log_2(\sigma) + 1.792 \quad (1)$$

The noise bits in each pixel are inherently incompressible, so the theoretical upper limit on the compression ratio for these images is given by $BitsPerPixel/N_{bits} = 16/N_{bits}$.

Figure 1 shows the measured compression ratios (original image size divided by the compressed image size) for the different compression algorithms when using the default image tiling pattern (i.e., row by row tiles for Rice, GZIP, and BZIP2, and 16 rows per tile for Hcompress). The compression ratios are plotted as a function of the equivalent number of noise bits in each image. As can be seen, the compression ratio achieved by BZIP2 is well below that of Rice and Hcompress. The primary reason for this is that the BZIP2 algorithm is not very efficient when compressing small files (e.g., the 500-pixel tiles in this case). The performance of BZIP2 improves significantly when compressing larger tiles, as shown in Figure 2, where the images were compressed as a single tile so that the tile contains 500 x 500 = 250000 pixels. Thus, BZIP2 can provide better compression than Rice, and even Hcompress, when compressing large blocks of data, however this is often not an effective tiling strategy because it can lead to severe I/O bottlenecks for applications that read the compressed image one row at a time.

In order to better quantify the effect of tile size on compression ratio, Table 1 shows the minimum tile size required by BZIP2 to produce the same amount of compression as the Rice algorithm (which is relatively insensitive of the effects of tile size) as a function of the number of noise bits in the image. This shows that BZIP2 is only competitive with Rice (or Hcompress) when the tiles contain at least several thousand pixels, which is larger than is usually the case when using the default row by row tiling pattern.

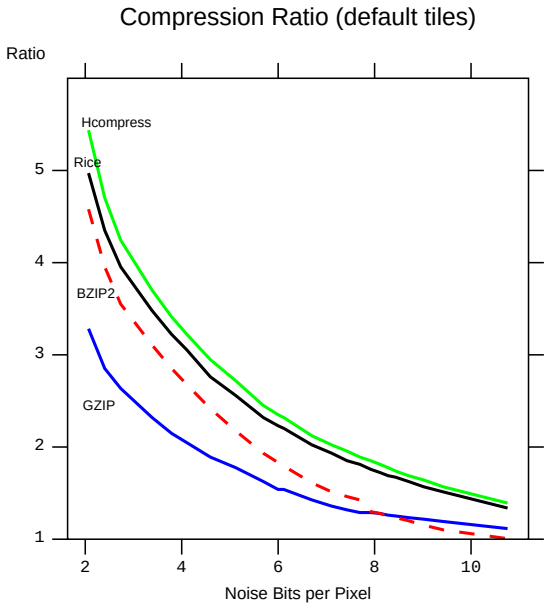

Fig. 1.— Compression ratios of the 4 algorithms when using the default image tiles. BZIP2 performs worse than Hcompress and Rice, but better than GZIP.

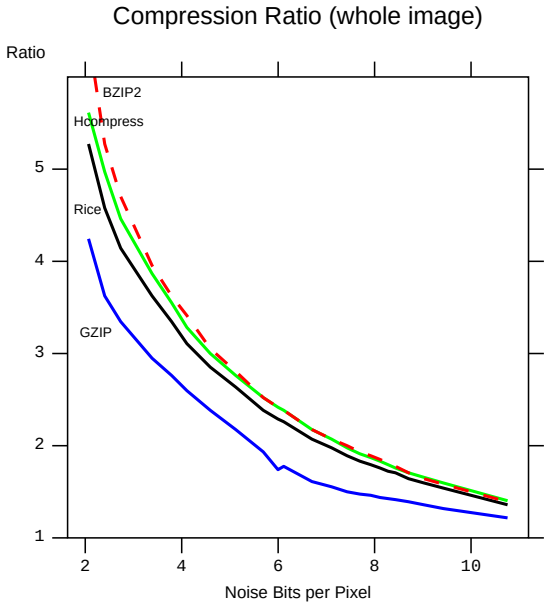

Fig. 2.— Compression ratios of the 4 algorithms when compressing the whole image as a single tile. BZIP2 now slightly exceeds the compression ratio of Hcompress for images with less than 5 bits of noise.

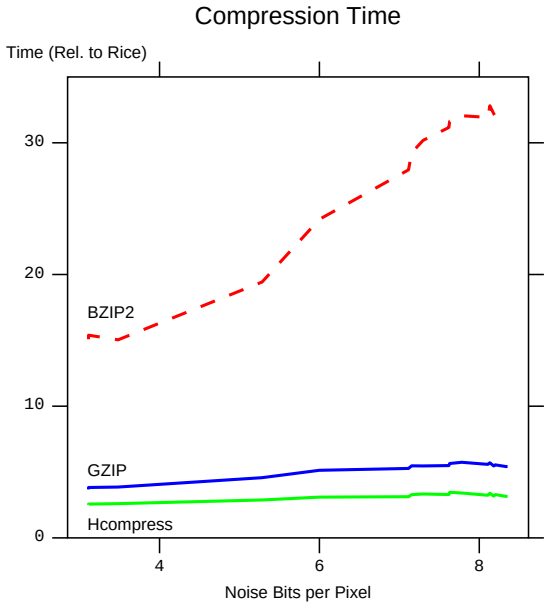

Fig. 3.— Image compression CPU times for Hcompress, GZIP, and BZIP2 relative to the compression time when using Rice, when using the default (row by row) image tiles.

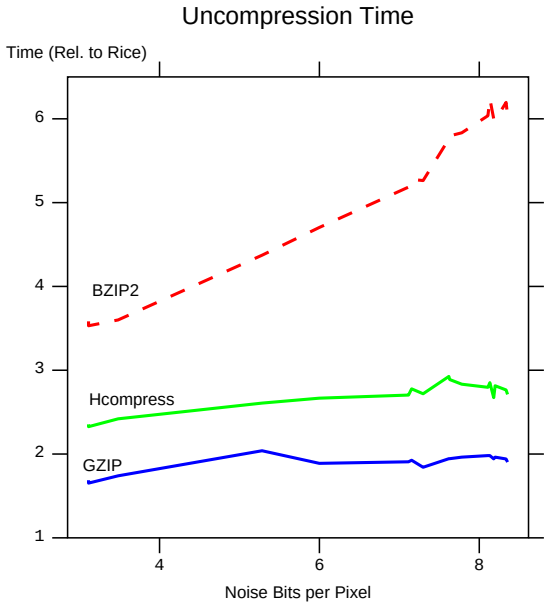

Fig. 4.— Image uncompression CPU times for Hcompress, GZIP, and BZIP2 relative to the uncompression time when using Rice, when using the default (row by row) image tiles.

3. Comparison of Compression and Uncompression Times

The relative speed of the different compression algorithms was measured on a sample of astronomical 16-bit integer CCD images, each 1112 x 4096 pixels in size, taken with the NOAO Mosaic camera. These larger-sized images were used in these timing tests in order to measure the compression and uncompression CPU times with greater precision. The equivalent number of noise bits per pixel in each image was empirically measured using the method described in Pence et al. (2009).

Figure 3 shows the relative image compression CPU times for the different algorithms plotted as a function of the number of noise bits per pixel in the image, when using the default tiling pattern. This plot shows the CPU time of the Hcompress, GZIP, and BZIP2 algorithms relative to that of the Rice algorithm (which is the fastest, and also has no dependence on the amount of noise in the image). BZIP2 is clearly much slower than all the other algorithms; it is 15 to 30 times slower than Rice, and 4 to 6 times slower than GZIP, depending on the amount of noise in the image.

Similarly, Figure 4 shows the relative image uncompression CPU times of the different algorithms. Here again, BZIP2 is much slower, requiring 4 to 6 times more CPU time than Rice, and 2 to 3 times more than GUNZIP.

These tests were repeated compressing the whole image as a single tile, but the relative speeds of the different algorithms did not change significantly (within a factor of 2).

4. Conclusions

Based on these tests there appears to be little advantage to using the BZIP2 algorithm within the FITS tiled image compression convention, as compared to the 3 currently supported general purpose algorithms, Rice, Hcompress, and GZIP. In particular, when using the recommended default row-by-row tiling pattern, the Rice algorithm provides greater compression than BZIP2 with 15 – 30 times faster compression speed and 4 – 6 times faster uncompression speed. The only case where BZIP2 provides better compression ratios than Rice or Hcompress is when compressing

Table 1. Minimum tile size needed for BZIP2 to equal the compression ratio of Rice.

Nbits	Pixels
4	2000
5	3000
7	8000

low-noise images (with less than 5 bits of noise per pixel) using large tile sizes (more than several thousand pixels). But this typically only provides about 10% greater compression than Hcompress, at the expense of much greater CPU times..

Consequently, at this time, public support for the BZIP2 algorithm will not be added to the CFITSIO library. The added burden of supporting BZIP2 in CFITSIO, and in other general FITS software, is not currently justified. It should be noted that the HDF5 Group reached a similar conclusion (HDF 2002) and opted to not officially support BZIP2 in their data format.

None the less, users may still take advantage of BZIP2 by externally compressing whole FITS files using the bzip2 command line program that is available on most machines. This may provide significantly better compression than when using the GZIP utility, and may be justified in situations where the additional CPU load is not an issue.

REFERENCES

HDF Group 2002, http://www.hdfgroup.uiuc.edu/papers/papers/bzip2/

Pence, W. D. 1999, in ASP Conf. Ser., Vol. 172, Astronomical Data Analysis Software and Systems VIII, eds. D. M. Mehringer, R. L. Plante, & D. A. Roberts (San Francisco: ASP), 487

Pence, W. D., White, R. L., Greenfield, P., & Tody, D. 2000, in ASP Conf. Ser., Vol. 216, Astronomical Data Analysis Software and Systems IX, eds. N. Manset, C. Veillet, & D. Crabtree (San Francisco: ASP), 551

Pence, W. D., Seaman, R., & White, R. L. 2009, PASP, 121, xxx (http://arxiv.org/abs/0903.2140)

Seward, J. 2007, http://www.bzip.org

Seaman, R., Pence, W. D., White, R., Dickinson, M., Valdes, F., & Zarate, N. 2007, in ASP Conf. Ser., Vol. 376, Astronomical Data Analysis Software and Systems XVI, eds. R. A. Shaw, R. Hill, & D. J. Bell (San Francisco: ASP), 483