SZ Tutorial Hands-on Guide

Version of February 3, 2018

SZ development team

Table of contents

SZ Tutorial Hands-on Guide

SZ development team

1 Introduction

SZ is developed by Mathematics and Computer Science (MCS) division at Argonne National Laboratory. It is an error-bounded lossy data compressor, supporting offline compression and online in-situ compression. It provides different ways to control compression errors on demand, including absolute error bound, relative error bound, peak signal-to-noise ratio (PSNR), etc. It supports both C and Fortran programming language and multiple I/O libraries/formats, such as HDF5, Adios and NetCDF. Note that the latest version of SZ is implemented based on our IPDPS17 paper [1] instead of IPDPS16 paper [2].

2 Download and Installation

Downloading SZ [1, 3, 4]:

```
    [user@host]$ git clone http://github.com/disheng222/SZ
```

Installating SZ:

```
    [user@host]$ ./configure --prefix=[INSTALL_DIR]
    [user@host]$ make;make install
```

Note:

1. The dynamic link and static link are named as lib**SZ**.so and lib**SZ**.a (uppercase).
2. Zlib is already integrated in the SZ package. If you link libSZ.so or libSZ.a, you also need linking libZlib.so or libZlib.a to your code.
3. The above installation does not support Fortran by default. If you want to enable Fortran, please use –enable-fortran option when running the command "configure –prefix=[INSTALL_DIR]".

3 Quick Start

3.1 Quick test of installation

Go to the example/ directory and run test.sh to see if it can successfully compress some sample data files with different dimensions and various data types (single-precision or double-precision) using an absolute error bound of 1E-4. If the maximum absolute errors printed on the screen are all no greater than 1E-4, the test is supposed to be successful.

```
[user@host]$ cd example/
[user@host]$ ./test.sh
```

```
============== testing compression and decompression of 1D array ==============
./testdouble_compress sz.config testdata/x86/testdouble_8_8_128.dat 8192
cfgFile=sz.config
[SZ] Reading SZ configuration file (sz.config) ...
timecost=0.006269
done
./testdouble_decompress sz.config testdata/x86/testdouble_8_8_128.dat.sz 8192
timecost=0.010932
done
Max absolute error = 9.861434847902028622E-05
Max relative error = 2.009658648889981253E-05
Max pw_relative err = 9.861434847902028622E-05
......
./testdouble_compress sz.config testdata/x86/testdouble_8_8_8_128.dat 8 8 8 128
cfgFile=sz.config
[SZ] Reading SZ configuration file (sz.config) ...
timecost=0.005040
done
./testdouble_decompress sz.config testdata/x86/testdouble_8_8_8_128.dat.sz 8 8 8 128
timecost=0.005727
done
Max absolute error = 9.977675269823294002E-05
Max relative error = 6.6517828359110580554E-05
Max pw_relative err = 9.961846503880877789E-05
```

3.2 Executable command: sz

You can use the executable command "sz" to do the compression and decompression simply. The input data file is in binary format. "sz -h" will print the help information.

```
Usage: sz <options>
Options:
* operation type:
        -z <compressed file>: the compression operation with an optionally
                specified output file. (the compressed file will be
                named as <input_file>.sz if not specified)
        -x <decompressed file>: the decompression operation with an optionally
                specified output file. (the decompressed file will be
                named as <cmpred_file>.out if not specified)
        -p: print meta data (configuration info)
        -h: print the help information
* data type:
        -f: single precision (float type)
        -d: double precision (double type)
* configuration file:
        -c <configuration file> : configuration file sz.config
* error control: (the error control parameters here will overwrite the setting
                in sz.config)
        -M <error bound mode> : 10 options as follows.
                ABS (absolute error bound)
                REL (value range based error bound
                ABS_AND_REL (using min{ABS, REL})
                ABS_OR_REL (using max{ABS, REL})
                PSNR (peak signal-to-noise ratio)
                PW_REL (point-wise relative error bound)
        -A <absolute error bound>: specifying absolute error bound
        -R <value_range based relative error bound>: specifying relative error bound
        -P <point-wise relative error bound>: specifying point-wise relative error bound
```

```
* input data file:
        -i <original data file> : original data file
        -s <compressed data file> : compressed data file in decompression
* output type of decompressed file:
        -b (by default) : decompressed file stored in binary format
        -t : decompreadded file stored in text format
        -T : pre-processing with Tucker Tensor Decomposition
* dimensions:
        -1 <nx> : dimension for 1D data such as data[nx]
        -2 <nx> <ny> : dimensions for 2D data such as data[ny][nx]
        -3 <nx> <ny> <nz> : dimensions for 3D data such as data[nz][ny][nx]
        -4 <nx> <ny> <nz> <np>: dimensions for 4D data such as data[np][nz][ny][nx]
* print compression results:
        -a : print compression results such as distortions
* examples:
        sz -z -f -c sz.config -i testdata/x86/testfloat_8_8_128.dat -3 8 8 128
        sz -z -f -c sz.config -M ABS -A 1E-3 -i testdata/x86/testfloat_8_8_128.dat \
                        -3 8 8 128
        sz -x -f -s testdata/x86/testfloat_8_8_128.dat.sz -3 8 8 128
        sz -x -f -s testdata/x86/testfloat_8_8_128.dat.sz \
                        -i testdata/x86/testfloat_8_8_128.dat -3 8 8 128 -a
        sz -z -d -c sz.config -i testdata/x86/testdouble_8_8_128.dat -3 8 8 128
        sz -x -d -s testdata/x86/testdouble_8_8_128.dat.sz -3 8 8 128
        sz -p -s testdata/x86/testdouble_8_8_128.dat.sz
```

Note:

- The error bounds could be set in the configuration file sz.config or using the options -M/-A/-R/-P in the command. The error controls in the command using -M/-A/-R/-P will overwrite the settings in the configuration file.
- Data types and other parameters can be set in the configuration file sz.config.
- -z and -x indicate 'compression' and 'decompression' respectively. They can also be used to specify the 'compressed data file' for the operation of compression and 'decompressed data file' for the operation of decompression. For instance, sz -z ./testdata.sz -i testdata/x86/testfloat_8_8_128.dat -3 8 8 128; sz -x ./testdata.sz.out -s ./testdata.sz -3 8 8 128

Types of Error bounds:

- absolute error bound (-M ABS): The absolute error bound (denoted δ) is a constant, such as 10^{-6} (a.k.a., 1E-6).
- relative error bound (-M REL): The relative error bound ratio (a.k.a., value-range based relative error bound).
- point-wise relative error bound (-M PW_REL): Different data points have different error bounds. the larger the value, the larger the absolute error bound for that data point.
- peak signal-to-noise ratio (-M PSNR): the larger the PSNR, the smaller the normalized mean squared error.

4 API

Most common APIs are listed as follows. More APIs can be found in the user guide.

- int SZ_Init(char *configFilePath); /*return SZ_SCES or SZ_NSCS*/
- unsigned char *SZ_compress(int dataType, void *data, size_t *outSize, size_t r5, size_t r4, size_t r3, size_t r2, size_t r1);
- void *SZ_decompress(int dataType, unsigned char *bytes, size_t byteLength, size_t r5, size_t r4, size_t r3, size_t r2, size_t r1);
- void SZ_Finalize();

We present an example to illustrate compression in the C code. The example code can be found in testfloat_compress.c in the directory example/.

```c
int main(int argc, char * argv[])
{
    size_t r5=0,r4=0,r3=0,r2=0,r1=0;

    /*Get the dimension information and set configuration file - cfgFile*/
    .....

    /*Initializing the compression environment by loading the configuration file.
   SZ_Init(null) will adopt default setting in the compression.*/
    int status = SZ_Init(cfgFile);
    if(status == SZ_NSCS)
    exit(0);
    sprintf(outputFilePath, "%s.sz", oriFilePath); //specify compression file path

    //read the binary data
    size_t nbEle;
    float *data = readFloatData(oriFilePath, &nbEle, &status);
    if(status != SZ_SCES)
    {
        printf("Error: data file %s cannot be read!\n", oriFilePath);
        exit(0);
    }

    /*Perform compression. r5, ...., r1 are sizes at each dimension. The size of a
    nonexistent dimension is 0. For instance, for a 3D dataset (10x20x30), the
   setting is r5 = 0, r4 = 0, r3 = 10, r2 = 20, r3 = 30. SZ_FLOAT indicates single
   -precision. */
    size_t outSize;
    unsigned char *bytes = SZ_compress(SZ_FLOAT, data, &outSize, r5, r4, r3, r2,
   r1);

    //write the compression bytes to 'outputFilePath'
    writeByteData(bytes, outSize, outputFilePath, &status);
    if(status != SZ_SCES)
    {
        printf("Error: data file %s cannot be written!\n", outputFilePath);
        exit(0);
    }

    /*Do not forget to free the memory of compressed data if they are not useful
   any more.*/
    free(bytes);
    free(data);
    SZ_Finalize();
    return 0;
}
```

We present an example to illustrate decompression in the C code, and it can be found in testfloat_decompress.c. Note that the decompression does not need SZ_Init(configFile).

```c
int main(int argc, char * argv[])
{
    size_t r5=0,r4=0,r3=0,r2=0,r1=0;
    size_t byteLength;
    ......

    /*read compressed data file*/
    int status;
    unsigned char *bytes = readByteData(zipFilePath, &byteLength, &status);
    if(status!=SZ_SCES)
    {
        printf("Error: %s cannot be read!\n", zipFilePath);
        exit(0);
    }

    /*Perform decompression*/
    float *data = SZ_decompress(SZ_FLOAT, bytes, byteLength, r5, r4, r3, r2, r1);
    free(bytes); /*free the memory of compressed data*/

    /*write decompressed data in bytes*/
    writeFloatData_inBytes(data, nbEle, outputFilePath, &status);
    if(status!=SZ_SCES)
    {
        printf("Error: %s cannot be written!\n", outputFilePath);
        exit(0);
    }

    free(data);/*free memory for decompressed data*/
    return 0;
}
```

5 Optimization of Compression Quality

In this section, we introduce how to optimize the compression quality based on specific user demand, by adjusting different parameter settings in the configuration file **sz.config**. The involved setting includes four parameters, quantization_intervals, max_quant_intervals, szMode, and gzipMode.

- *quantization_intervals*: it has two options (a positive integer number or 0). If it is set to a positive number such as 65536, the compression will adopt such a specific number of bins in the linear-scaling quantization step of the compression. If it is set to 0, SZ will optimize the the number of quantization bins based on max_quant_intervals. In general, we recommend to set quantization_intervals to 0.
- *max_quant_intervals*: This parameter is valid only when quantization_intervals = 0. In this case, SZ will estimate the best-fit number of quantization bins based on the maximum number of bins (specified by max_quant_intervals), such that 99% of data points are predictable during the compression.
- *szMode*: SZ_BEST_SPEED (without Gzip) or SZ_BEST_COMPRSSION (with Gzip)
- *gzipMode*: Gzip_BEST_SPEED, Gzip_DEFAULT_COMPRSSION or Gzip_BEST_COMPRSSION

Based on our experience, we recommend the following settings based on your demand.

- **BEST_CMPR_hard_sett**: If you want to get the best compression ratio, while the data set is very large in size (such as 5GB+) and very hard to compress or the error bound is set to pretty small (such as 1E-8), the recommended setting is as follows. Note that we recommend using szMode = SZ_BEST_SPEED, because for hard-to-compress cases, we observe that SZ_BEST_SPEED and SZ_BEST_COMPRESSION lead to very similar compression ratio in general.

```
quantization_intervals = 0
max_quant_intervals = 4194304
szMode = SZ_BEST_SPEED
gzipMode = Gzip_BEST_SPEED
```

- **BEST_CMPR_easy_sett**: If you want to get the best compression ratio, and the data set is easy to compress or the error bound is relatively high, the recommended setting is as follows. Note that we recommend setting szMode to SZ_BEST_COMPRESSION because based on our experience, SZ_BEST_COMPRESSION is important for easy-to-compress cases.

```
quantization_intervals = 0
max_quant_intervals = 65536
szMode = SZ_BEST_COMPRESSION
gzipMode = Gzip_BEST_SPEED
```

- **GOOD_RATE_general_sett**: If you want to have a fast compression rate with a good compression ratio, the recommended setting is as follows.

```
quantization_intervals = 0
max_quant_intervals = 65536
szMode = SZ_BEST_SPEED
gzipMode = Gzip_BEST_SPEED
```

- **BEST_RATE_general_sett**: If you want to have the fastest compression rate, the recommended setting is as follows.

```
quantization_intervals = 65536
max_quant_intervals = 65536 #to be ignored since quantization_intervals > 0
szMode = SZ_BEST_SPEED
gzipMode = Gzip_BEST_SPEED
```

The following table presents the compression ratios and compression rate, using different settings. Based on the table, we can see that BEST_RATE_general_sett always leads to the best compression speed (i.e., highest compression rate). BEST_CMPR_easy_sett leads to the best compression ratio in this case. Note that this data set is very small in size (it is only 100MB), so BEST_CMPR_hard_sett's compression ratio is not as high as that of BEST_CMPR_easy_sett, because of overhead of Huffman tree. If the data size is very large and the error bound is set very small, BEST_CMPR_easy_sett is the recommended setting.

Table 1: Evaluation using different settings (Hurricane dataset: Uf01.dat (100x500x500,single-precision)

	abs_err=1E-8		**abs_err=1E-4**	
	compression ratio	**compression time**	**compression ratio**	**compression time**
BEST_CMPR_hard_sett	1.52	2.503 sec	5.96	1.236 sec
BEST_CMPR_easy_sett	1.792	3.08 sec	6.05	1.689 sec
GOOD_RATE_general_sett	1.70	1.519 sec	5.96	1.194 sec
BEST_RATE_general_sett	1.704	1.346 sec	6	0.953 sec

6 Exercises

1. Compress CLDLOW.dat in CESM-ATM-Tylor, with value_range based relative error bound = 0.0001, targeting the best compression ratio.
2. Compress xx.dat in HACC, using value_range based relative error bound = 0.001, targeting the best compression ratio.
3. Compress vx.dat in HACC, using point-wise relative error bound = 0.001, targeting the best compression ratio.

References

[1] D. Tao, S. Di, Z. Chen, and F. Cappello, “Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization,” in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1129–1139, May 2017.

[2] S. Di and F. Cappello, “Fast error-bounded lossy HPC data compression with SZ,” in *IPDPS 2016*, pp. 730–739, 2016.

[3] S. Di, D. Tao, and F. Cappello. SZ (github website): http://github.com/disheng222/SZ. Online.

[4] S. Di, D. Tao, and F. Cappello. SZ (Argonne website): https://collab.cels.anl.gov/display/ESR/SZ. Online.