Fast Error-Bounded HPC Data Compressor (SZ-0.5.11)

User Guide (Version 0.5.11)

Mathematics and Computer Science (MCS)
Argonne National Laboratry
Contact: Sheng Di (sdi1@anl.gov)
Dec., 2015

Table of Contents

1. Brief description

- **SZ** (short for ***Squeeze***) is an **error-bounded** HPC in-situ data compressor for significantly reducing the data sizes, which can be leveraged to improve the checkpoint/restart performance and post-processing efficiency for HPC executions.
- SZ can be used to compress different types of data (single-precision and double-precision) and any shapes of the array. Current version supports up to five dimensions. Higher dimensions can also be extended easily.
- SZ is very easy to use. It supports three programming languages: Fortran, C and Java.
- SZ supports many different architectures, including x86_32bits (denoted by linux_x86 in the Makefile), x86_64bits (denoted by linux_x64 in the Makefile), ARM (denoted by linux_arm), SOLARIS (denoted by solaris), IBM BlueGene series (denoted by pps).

- SZ allows setting the compression error bound based on *absolute error bound* and/or *relative error bound*, by using sz.config (which can be found in the directory *example*) or by passing arguments through programming interfaces.
  - **Absolute error bound** (namely *absErrBound* in the configuration file sz.config): It is to limit the (de)compression errors to be within an absolute error. For example, absErrBound=0.0001 means the decompressed value must be in [V-0.0001,V+0.0001], where V is the original true value.
  - **Relative error bound** (called *relBoundRatio* in the configuration file sz.config): It is to limit the (de)compression errors by considering the global data value range size (i.e., taking into account the range size (max_value - min_value)). For example, suppose relBoundRatio is set to 0.01, and the data set is {100,101,102,103,104,...,110}. That is, the maximum value is 110 and minimum value is 100. So, the global value range size is 110-100=10, and the error bound will actually be 10*0.01=0.1, from the perspective of "relBoundRatio".
- Users can set the real compression error bound based on only absErrorBound, relBoundRatio, or a kind of combination of them. Two types of combinations are provided: **AND, OR**. ABS_**AND**_REL means that both of the two bounds (absErrorBound and relBoundRatio) will be considered in the compression. ABS_**OR**_REL means that the compression error is satisfied as long as one type of bound is met.

2. How to install SZ

The SZ software can be downloaded from http://collab.mcs.anl.gov/display/ESR/SZ

Perform the following three simple steps to finish the installation:

a) Set environment vairable JAVAHOME path. (HINT: edit ~/.bashrc by adding this line "export JAVAHOME=[JAVA_INSTALL_PATH]" and adding this line "export PATH=$JAVAHOME/bin:$PATH", and then execute "source ~/.bashrc" to validate the modification)

b) Configure [SZ_INSTALL_PATH]/Makefile (such as /home/sdi/SZ_0.5.11/Makefile), by modifying three parameters, SZPATH, JAVAHOME, and ARCHITECTURE.
(SZPATH is the installation path, and ARCHITECTURE is the processor's architecture, such as linux_x86, linux_x64, linux_arm, solaris, pps).

For example,

SZPATH	= /home/sdi/sz-0.5.11
JAVAHOME	= /home/sdi/jdk1.8
ARCHITECTURE	= linux_x64

(Note: JDK version must be 1.7 or higher)

c) Go to the [SZPATH], run the following commands:

make
make install

3. Quick Start

The testing cases can be found in **[SZ_ROOT_PATH]/example**
Modify the three parameters in [SZ_ROOT_PATH]/example/configure as follows:

```
##     PLEASE SET THIS VARIABLE BEFORE COMPILING
SZPATH              = [SZ_INSTALL_PATH]
JAVAHOME            = [JDK_HOME_PATH] (version must be no less than 1.7)
#ARCHITECTURE is set to linux_x64,linux_x86,linux_arm,solaris, or ppc
#ppc means IBM PowerPC (such as IBM BlueGene series)
ARCHITECTURE    = linux_x64
```

Then, execute "./configure" make the configuration.
Then, compile the testing cases by executing "make" in [SZ_ROOT_PATH]/example, and the examples are ready to use now.
(**A minor note**: Each testing command needs to run with the configuration file sz.config. The first line of the sz.config is SZ_CLASSPATH, which should point to the java_class_path of sz.jar. In fact, SZ_CLASSPATH is automatically filled when you compile the SZ examples by running "make" in the example directory. You don't have to modify it manually, but if you changed the path of sz.jar manually, you should also change SZ_CLASSPATH in sz.config accordingly.)

3.1 Compression

Testing commands:
Run "**./testdouble_compress sz.config teestdata/x86/testdouble_8_8_128.dat 8 8 128**" to compress the data testdouble_8_8_128.dat.
Run "**./testdouble_compress sz.config teestdata/x86/testdouble_8_8_8_128.dat 8 8 8 128**" to compress the data testdouble_8_8_8_128.dat.
Run "**./testfloat_compress sz.config teestdata/x86/testfloat_8_8_128.dat 8 8 128**" to compress the data testfloat_8_8_128.dat

Remark:
testdouble_8_8_128.dat and testdouble_8_8_8_128.dat are two binary testing files, which contain a 3d array (128X8X8) and a 4d array (128X8X8X8) respectively. Their data values are shown in the two plain text files, testdouble_8_8_128.txt and testdouble_8_8_8_128.txt. These two data files are from FLASH_Blast2 and FLASH_MacLaurin respectively (the two test data are both extracted at time step 100). The compressed data files to be generated

are named testdouble_8_8_128.dat.sz and testdouble_8_8_8_128.dat.sz respectively.

./testfloat_compress.c is an example to show how to compress single-precision data. Use test**float**_8_8_128.dat as the input when testing the compression of single-precision data.

sz.config is the configuration file. The key settings are *errorBoundMode*, *absErrBound*, and *relBoundRatio*, which are described below.

- ***absErrBound*** refers to the absolute error bound, which is to limit the (de)compression errors to be within an absolute error. For example, absErrBound=0.0001 means the decompressed value must be in [V-0.0001,V+0.0001], where V is the original true value.
- ***relBoundRatio*** refers to relative bound ratio, which is to limit the (de)compression errors by considering the global data value range size (i.e., taking into account the range size (max_value - min_value)). For example, suppose relBoundRatio is set to 0.01, and the data set is {100,101,102,103,104,...,110}. In this case, the maximum value is 110 and the minimum is 110. So, the global value range size is 110-100=10, and the error bound will be 10*0.01=0.1, from the perspective of "relBoundRatio".
- ***errorBoundMode*** is to define a combination of the above two types of error bounds. There are four types of values: **ABS, REL, ABS_AND_REL, ABS_OR_REL**,.
  - **ABS** takes only "absolute error bound" into account. That is, relative bound ratio will be ignored.
  - **REL** takes only "relative bound ratio" into account. That is, absolute error bound will be ignored.
  - **ABS_AND_REL** takes both of the two bounds into account. The compression errors will be limited using both absErrBound and relBoundRatio*rangesize. That is, the two bounds must be both met.
  - **ABS_OR_REL** takes both of the two bounds into account. The compression errors will be limited using either absErrBound or relBoundRatio*rangesize. That is, only one bound is required to be met.

sz.config is the configuration file used to set the compression environment. Please read the comment in the file to understand the parameters.

3.2 Decompression

Testing commands:
./testdouble_decompress sz.config testdouble_8_8_128.dat.sz
./testdouble_decompress sz.config testdouble_8_8_8_128.dat.sz
./testfloat_decompress sz.config testfloat_8_8_128.dat.sz

Remark:

- Unlike compression, you don't have to provide the error bound information (such as errBoundMode, absErrBound, and relBoundRatio), when performing the data

decompression, because such information is stored in the compressed data stream.

- The output files of the test_decompress.c are .out files, i.e., testdouble_8_8_128.dat.sz.out and testdouble_8_8_8_128.dat.sz.out respectively. You can compare .txt file and .out file for checking the compression errors for each data point. For instance, compare testdouble_8_8_8_128.txt and testdouble_8_8_8_128.dat.sz.out.

4. Application Programming Interface (API)

Programming interfaces are provided in three programming languages – C, Fortran, and Java. The usage methods of the interfaces are quite similar across different programming languages, with only a few differences. For example, In C interface, a *dataType* (either SZ_FLOAT or SZ_DOUBLE) is required, while Fortran interface does not require this argument because of the function overloading feature.

4.1 Compression/Decompression by C Interfaces

There are three key interfaces for compression/decompression in C.

(1) Initialize the compressor by calling SZ_Init();

(2) Compress the data (a floating-point array) by SZ_compress(), or decompress the data by SZ_decompress();

(3) Finalize the compressor by SZ_Finalize() if the compressor won't be used any more.

Interfaces:

(a) ***SZ_Init***

Initialize the SZ compressor. SZ_Init() just needs to be called only **once** before performing multiple compressions for different variables (data arrays).

Synopsis: `void SZ_Init(char *configFilePath);`

Input:

configFilePath the configuration file path (such as example/sz.config)

Return: 1 (failure) or 0 (success)

(b) ***SZ_compress***

Compress the floating-point data array. Two types of interfaces are provided, as shown below. For the interface *SZ_compress* and *SZ_compress_rev*, the three important control parameters (errBoundMode, absErrBound, and relBoundRatio) will be given by the configuration file sz.config. For the rest interfaces, the three control parameters will be passed using arguments and the parameter settings in the sz.config will be ignored in this case.

There are six compression interfaces with different arguments, as listed below. The user

just needs to choose one of them in compressing data.

Synopsis:

- `char *SZ_compress(int dataType, void *data, int *outSize, int r5, int r4, int r3, int r2, int r1);`
- `char *SZ_compress_args(int dataType, void *data, int *outSize, int errBoundMode, double absErrBound, double relBoundRatio, int r5, int r4, int r3, int r2, int r1);`
- `int SZ_compress_args2(int dataType, void *data, char* compressed_bytes, int *outSize, int errBoundMode, double absErrBound, double relBoundRatio, int r5, int r4, int r3, int r2, int r1);`
- `char *SZ_compress_rev(int dataType, void *data, void *reservedValue, int *outSize, int r5, int r4, int r3, int r2, int r1);`
- `char *SZ_compress_rev_args(int dataType, void *data, void *reservedValue, int *outSize, int errBoundMode, double absErrBound, double relBoundRatio, int r5, int r4, int r3, int r2, int r1);`
- `int SZ_compress_rev_args2(int dataType, void *data, void *reservedValue, char* compressed_bytes, int *outSize, int errBoundMode, double absErrBound, double relBoundRatio, int r5, int r4, int r3, int r2, int r1);`

Input:

dataType	the indicator that indicates the data type (two options: either *SZ_FLOAT* or *SZ_DOUBLE*)
data	the variable that contains the data to be compressed.
reservedValue	the variable that indicates a reserved value. (Some scientific simulations such as weather simulation has many "default" reserved value like 1.0e+36 in the snapshot data. In this case, users can set the reservedValue using this argument)
compressed_bytes	the address that contains the compressed bytes
outSize	the data stream size (in bytes) after compression.
r5	size of dimension 5
r4	size of dimension 4
r3	size of dimension 3
r2	size of dimension 2
r1	size of dimension 1

Return: Compressed data stream (in the form of bytes)

Usage tips: The dimension of the variable is determined based on the five dimension parameters (r5, r4, r3, r2, and r1). For instance, if the variable is a 2D array (M X N), then r5=0, r4=0, r3=0, r2=M, and r1=N. If the variable to protect is a 4D array, then only r5 is set to 0. (See test_compress.c for details).

(c) *SZ_decompress*

Decompress/recover the data. Two options, as listed below.

Synopsis:

- **void *SZ_decompress(int dataType, char *bytes, int byteLength,**
  int r5, int r4, int r3, int r2, int r1);
- **int SZ_decompress_args(int dataType, char *bytes, int byteLength,**
  void* decompressed_array,
  int r5, int r4, int r3, int r2, int r1);

Input:

dataType	the indicator to indicate the data type (either *SZ_FLOAT* or *SZ_DOUBLE*)
bytes	the compressed data stream to be decompressed
byteLength	length of the compressed data stream
decompressed_array	the address to store decompressed data
r5	size of dimension 5
r4	size of dimension 4
r3	size of dimension 3
r2	size of dimension 2
r1	size of dimension 1

Return: the recovered data array decompressed from the compressed bytes.

(d) ***SZ_Finalize***

Release the memory and compression environment

Synopsis: **int SZ_Finalize();**

Input: none.

Return: none.

4.2 Compression/Decompression by Fortran Interfaces

Interfaces:

(a) ***SZ_Init***

Initialize the SZ compressor. SZ_Init() just needs to be called only **once** before performing multiple compressions for different variables (data arrays).

Synopsis: **SZ_Init(configFilePath, ierr);**

Input:

configFilePath	configuration file path (e.g., sz.config) **CHARACTER(len=32) :: configFilePath**

Output:

ierr	successful (0) or failed (1) **INTEGER(Kind=4) :: ierr**

(b) ***SZ_Compress***

Compress the floating-point data array. Two types of interfaces are provided, as shown below. For the first one, the three important control parameters (errBoundMode, absErrBound, and relBoundRatio) will be given by the configuration file sz.config. For

the second one, the three control parameters will be passed using arguments, so in this case, the parameter settings in the sz.config will be ignored.

Synopsis A:

SZ_compress(data, bytes, outSize);

SZ_compress(data, reservedValue, bytes, outSize);

Input:

data the data array to be compressed
(the data here is a floating-point data array with up to 5 dimensions. For example, "REAL(KIND=8), DIMENSION(:,:,:) :: *data*" indicates a 3D double-precision array, where *data* refers to the array variable.)

reservedValue the reservedValue

Output:

bytes the byte stream generated after the compression
INTEGER(kind=1), DIMENSION(:), allocatable :: bytes

outsize the size (in bytes) of the byte stream
INTEGER(kind=4) :: OutSize

Synopsis B:

SZ_Compress (data, bytes, outSize,
errBoundMode, absErrBound, relBoundRatio);

SZ_Compress (data, reservedValue, bytes, outSize,
errBoundMode, absErrBound, relBoundRatio);

Input:

data the data array to be compressed
(the data here is a floating-point data array with up to 5 dimensions. For example, "REAL(KIND=8), DIMENSION(:,:,:) :: *data*" indicates a 3D double-precision array, where *data* refers to the array variable.)

reservedValue the reservedValue

errBoundMode the error bound mode.
Four options: ABS, REL, ABS_AND_REL, ABS_OR_REL
INTEGER(kind=4) :: ErrBoundMode

absErrBound absolute error bound
REAL(kind=4 or 8) :: absErrBound

relBoundRatio relative bound ratio
REAL(kind=4 or 8) :: relBoundRatio
(Details about error bound mode, absolute error bound, and relative bound ratio can be found in Section 3.1)

Output:

bytes the byte stream generated after the compression
INTEGER(kind=1), DIMENSION(:), allocatable :: bytes

outsize the size (in bytes) of the byte stream

INTEGER(kind=4) :: OutSize

(c) ***SZ_Decompress***

Decompress/recover the data

Synopsis:

SZ_Decompress(bytes, data, [r1,r2,…])

Input:

bytes	the compressed data stream to be decompressed
	INTEGER(kind=1), DIMENSION(:) :: Bytes
data	length of the compressed data stream
	REAL(KIND=4 or 8), DIMENSION(:,:,…:,:), allocatable :: data
r1	size of dimension 1
r2	size of dimension 2
r3	size of dimension 3
r4	size of dimension 4
r5	size of dimension 5
	INTEGER(kind=4) :: r1[, r2, r3, r4, r5]

Usage tips: SZ_Decompress supports the decompression of the array with at most 5 dimensions. The dimension sizes (such as r1, r2, ….) are supposed to be provided. For example, in order to decompress a binary stream whose original data is a 3D array (r3=10,r2=8,r1=8), the function is like "SZ_Decompress(bytes, data, 8, 8, 10).

(d) ***SZ_Finalize***

Release the memory and compression environment

Synopsis: **SZ_Finalize();**

Input: none.

Return: none.

4.3 Compression/Decompression by Java Interfaces

Java interfaces are similar to C interfaces. The only two differences are:

(1) Java program doesn't require SZ_Finalize interface to terminate the compression environment.

(2) SZ_init() is not necessary either, unless you want to use sz.config to load the compression control parameters, such as errBoundMode, absErrBound, and relBoundRatio.

Class name: compression.SingleFillingCurveCompressVariable

Methods:

```
public static void SZ_Init(String configFilePath)
public static byte[] SZ_compress(float[] doubleData,
                                 int r5, int r4, int r3, int r2, int r1)

public static byte[] SZ_compress(float[] doubleData,
                 int errBoundMode, float absErrBound, float relBoundRatio,
```

int r5, int r4, int r3, int r2, int r1)

public static byte[] SZ_compress(double[] doubleData,
int r5, int r4, int r3, int r2, int r1)

public static byte[] SZ_compress(double[] doubleData,
int errBoundMode, double absErrBound, double relBoundRatio,
int r5, int r4, int r3, int r2, int r1)

Class name: compression.SingleFillingCurveDecompressVariable
Methods:
public static Object SZ_decompress(int dataType, byte[] bytes, int r5, int r4, int r3, int r2, int r1)
(**Note:** dataType is either 0 or 1. 0 indicates float type, while 1 indicates double type. The returned type is either float[] or double[], depending on the dataType)

5 Test cases

example/testdouble_compress.c
example/testdouble_decompress.c
example/testfloat_compress.c
example/testfloat_decompress.c
example/testdouble_compress.f90
example/testdouble/decompress.f90

6. Version history

The latest version (version 0.5.11) is the recommended one.
Version 0.1-0.4 were used in the internal tests.
The latest version also allows users to set the maximum memory usage to deal with large data set. For example, if the data to be compressed is very huge, you need to increase the MAX_HEAP in the sz.config. It also supports different processor architectures.

7. Q&A and Trouble shooting

1. Do I need to call SZ_init() every time I compress a variable in the program?
Answer: No. In a program (or one simulation), SZ_init() just needs to be called once at the beginning, and thereafter you can always compress different variables using the compression/decompression functions on demand, until SZ_finalize() is called.

2. If I want to use SZ_compress_args() function and specify the errorBoundMode and bounds at run time instead of using the sz.config, do I need to call SZ_init()?
Answer: Yes. SZ_init() is an initialization step, which initializes the environment for the compressor. It has to be called anyway before calling other compression/decompression functions. It just needs to be called once for each simulation.

3. How to deal with "Error: The input file or data stream is not in SZ format!"?
Answer: This error is because the input file or data stream used to be decompressed is probably not the byte steam compressed/generated by the SZ. Please use the compressed file (ending with .sz) in the decompression.

4. I encountered a core dump error, and the log file shows such an Exception: "OutofMemoryError"
Answer: Increase the value of MAX_HEAP in sz.config. MAX_HEAP is the maximum heap memory that can be allocated to the compressor in the execution for purpose of safety. Note that you can set a relatively large number (such as 2048m means 2GB and 8096m means 8GB). The compression process will still use on-demand memory at runtime, instead of always using up all heap memory size.

5. Too large compression errors? Some extremely large data values cannot be decompressed to the original values. For example: The decompressed value of 1.0E+36 is 99999996169031624536541560020821606 4. Is this incorrect?
Answer:
Both of the above results are correct.
In fact, 1.0E+36 and 99999996169031624536541560020821606 4 have exactly the same IEEE 754 representation, which is 0x7B4097CE = 01111011 01000000 10010111 11001110. The users are supposed to be aware of such a possible loss of the floating-point numbers because of the machines.

<END>