Performance evaluation report:
gzip, bzip2 compression with and without shuffling algorithm

0. Purposes:

- To evaluate the performance of the combination of shuffling and general compression packages such as gzip and bzip2
- To evaluate the possibility of integrating shuffling algorithm to HDF5

1. What's the shuffling algorithm and why it is useful for data compression?

The shuffling algorithm itself will not compress the data; it is only changing the byte order in the data stream. Since all scientific data more or less have locality; that means many numbers are very close to each other; so when we use the combination of shuffling algorithm with general compression packages, we may take advantage of this characteristic of data and obtain better compression ratio.

An example:

We have five 32-bit unsigned integers: 1, 23, 43, 56, 35
The hexidecimal form of these numbers is 0x01, 0x17, 0x2B, 0x38, 0x23

In big-endian machine, these numbers are stored in memory as follows:

```
0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x17 0x00 0x00 0x00 0x2B 0x00 0x00
0x00 0x38 0x00 0x00 0x00 0x23
```

The shuffling algorithm re-arrange the byte order of these numbers, put the first byte of every number in the first chunk and then the second byte of every number and so on. After shuffling the data stream, in memory these numbers are stored as follows:

```
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x01 0x17 0x2B 0x38 0x23
```

You can see 15 continuous zeroes in the second data stream and all non-zero numbers are at the end. Imagine if we have 256 MB such kind of data and we may have 192 MB continuous data filling with 0 at this 256 MB buffer. Compression packages like gzip and bzip2 can greatly improve the compression ratio with this shuffling data set.

When reading the shuffled compressed data back to the memory, we should re-shuffling the data after decompressing the data.

The side effect of shuffling algorithm is that it may take extra time to shuffle and re-shuffle the data.

2. How the shuffling algorithm is integrated into HDF5

Shuffling algorithm is treated as a filter in the HDF5 I/O pipeline. To integrate the shuffling algorithm into the HDF5 includes the following steps:

a) Define a filter constant called H5Z_FILTER_SHUFFLE at H5Zpublic.h
b) Following the deflate example, declare a filter function called H5Z_filter_shuffle with the parameter set exactly as H5Z_filter_deflate.
c) Create a new source code file called H5Zshuffle.c, put the implementation of shuffling algorithm inside.
d) Define a new property list function called H5Pset_shuffle following the definition of H5Pset_deflate.
e) Following the function H5Pset_deflate, put the implementation of H5Pset_shuffle inside function H5Pdcpl.c.
f) At Makefile.in, add H5Zshuffle.c in the source file list.

3. Definitions in this report:

1) ***Compression ratio***: The ratio of the compressed file size or array size to the original file size or array size.

2) ***Encoding time of the library***: Difference of the elapsed time between writing an HDF5 dataset with compression and without compression

3) ***Decoding time of the library***: Difference of the elapsed time between reading an HDF5 dataset with compression and without compression

4) ***Improvement of compression ratio***: The subtraction of the compression ratio of the data array without data shuffling and re-shuffling to the compression ratio of the data array with data shuffling

5) ***Difference of unit encoding time:*** The subtraction of total elapsed time of the 1MB data with data shuffling and compression to the encoding time of the1MB data with compression only.

6) ***Difference of unit decoding time:*** The subtraction of total elapsed time of the 1MB data with data shuffling and decompression to the decoding time of the 1MB data with decompression only.

7) ***Relative overhead of encoding time with the addition of shuffling***: The ratio of extra processing time with the addition of shuffling in encoding stage to the encoding time without shuffling in the compression package

For example:
To encode a 1MB 32-bit floating point HDF5 data with gzip takes 0.1 second. When we shuffle the data and then use gzip compression, it takes 0.11 second. Relative overhead of encoding time with the addition of shuffling is (0.11-0.1)/0.11=9%.

8) ***Relative overhead of decoding time with the addition of shuffling***: The ratio of "extra" processing time used with the addition of shuffling in decoding to the decoding time without shuffling in the compression package

For example:
To decode a 1MB 32-bit floating point HDF5 data set with gzip takes 0.01 second. When we use gzip to decompress and then shuffle the data, it takes 0.012second. Relative overhead of encoding time with the addition of shuffling is (0.112-0.01)/0.01=20%.

9) ***Relative improvement of compression ratio:*** The ratio of improvement of compression ratio to the compressed percentage of the array without data shuffling

For example, the compression ratio for array A without shuffling is 0.8.
So the compressed percentage of the array is 1 – 0.8 = 0.2.
The compression ratio for array A with shuffling is 0.5.
So the improvement of compression ratio is 0.8-0.5 = 0.3.
Relative improvement of compression ratio is 0.3/0.2 = 1.5.

Note:

- gettimeofday is used to calculate encoding and decoding time of the library

4. Data collections

I totally collected 24 HDF5 datasets from HDF4 and HDF5 files. Among them there are 5 SAF datasets, 2 Swede radar datasets, 2 MIT physics datasets, 1 Spot dataset, 1 SWARM dataset, 13 NASA EOS datasets.

Table 1: Dataset information of the study

File Name	Dataset Name	Array size (byte)	Data type
TRIM	/DATA_GRANULE/SwathData/geolocation	4968704	Float32
TRIM	/DATA_GRANULE/SwathData/lowResCh	4347616	Int16
CERES1	/CERES_ES8_subset/Data Fields/CERES LW flux at TOA	5359200	Float32
ASTER2	/SurfaceReflectanceSWIR/Data Fields/Band5	10458000	Unsigned int16
ASTER1	/VNIR/VNIR_Swath/Data Fields/ImageData3B	22908000	Unsigned char
MODIS1	/MODIS_SWATH_Type_L1B/Data Fields/EV_500_RefSB	109944800	Unsigned int16
MODIS1	/MODIS_SWATH_Type_L1B/Geolocation Fields/Latitude	10994480	Float32
MODIS2	/mod07/Data Fields/Brightness_Temperature	2630880	Int16
MODIS3	/Standard_Map	2097152	Float32
MODIS4	/MODIS_Swath_Type_GEO/Data Fields/Height	5497240	Int16
CERES2	/Footprint Imager Radiance Statistics/5th percentile of imager radiances over full CERES FOV	572900	Float32
SRB	/Data-Set-11	817904	Float32
MOPS1	/MOP01/Data Fields/MOPITT Radiances	47142400	Float32
MIT1	/x.i	658560	Float64
MIT2	/x.i	1393200	Float64
SWARM	/inp_gammaFN-002000_gammaTT-000100_agentTTCash-000100_agentFNCash-005000/position/AgentFNImpl-E0-0	4000000	Float64
SPOT	/PIXEL DATA	146684610	Unsigned int16
swe-radar1	/v0	1440000	Float64
swe-radar2	/data	3787560	Float32
SAF1	/fld_ss_[1040…1103]	10250240	Float64
SAF2	/fld_velocity_[2012…2075]	35551488	Float64
SAF3	/ssrel__[0064…0127]	5125120	Int32
SAF4	/toporel__[0192…0255]	41000960	Int32
SAF5	/fld_coords_[0452…0515]	35551488	Float64

5. System descriptions

Linux 2.2.18smp i686
Physical memory 960 MB

CPU:
Two processors
Model name: Pentium III(Katmai)
Speed: 551.261 MHZ
Cache_size: 512KB

6. Experiment procedures

a) Converting HDF4 file to HDF5 file
All NASA data are in HDF4 format; so we are using h4toh5 utility to convert HDF4 data to HDF5 data.
b) Test the validity of shuffling algorithm
Since shuffling 8-bit data will not change the real data stream, so for 8-bit data, the data should be exactly the same with shuffling and without shuffling. ASTER1 data is used to check whether this is true.
c) Then bzip2 and gzip are added to I/O pipeline after the shuffling filter. We choose three levels: 1,6,9 in this performance study.
d) To do the performance comparison, we then calculate the improvement of compression ratio, relative overhead of decoding time with the addition of shuffling, relative overhead of encoding time with the addition of shuffling and generate charts.

7. Results
a) Control test
CR: Compression Ratio CL: Compression Level
File name: ASTER1 Array size: 22908000 bytes Data type: unsigned char

Table 1: Difference of compression ratio between shuffled run and un-shuffled run

CL	Un-shuffled Compressed Size (byte)	Shuffled Compressed Size (byte)	Un-shuffled CR	Shuffled CR	CR difference
bzip2 1	10110257	10110257	0.441341758	0.441341758	0
gzip 1	13577118	13577118	0.592680199	0.592680199	0
bzip2 6	9801662	9801662	0.4278707	0.4278707	0
gzip 6	13429576	13429576	0.586239567	0.586239567	0
bzip2 9	9758766	9758766	0.425998167	0.425998167	0
gzip 9	13412551	13412551	0.585496377	0.585496377	0

This test verifies that shuffling algorithm is implemented correctly and returns the correct output.

b) Results

- The combination of shuffling algorithm with bzip2 and gzip can gain improvement of compression ratio for most 32-bit and 64-bit data samples.
- On average, the improvement of compression ratio for float32 is 10% for both compression packages.
- On average, the improvement of compression ratio for float64 is 5% for both compression packages.
- Most cases show than less encoding time and decoding time are used for compression with the shuffling and bzip2.
- Most cases show than insignificant extra encoding and decoding time are used for compression with the shuffling and gzip.
- The combination of shuffling algorithm with bzip2 and gzip cannot significantly benefit for those data that can gain better compression ratio with bzip2 and gzip only.
- The combination of shuffling algorithm with bzip2 and gzip is generally not good for 16-bit data.

The following figures will show illustration of CR improvement, relative overhead of encoding time and decoding time of float32 and float 64 in detail.

X-axis: the number of data arrays
Y-axis:
The relative overhead of encoding and decoding with shuffling and bzip2/gzip.
The relative overhead of CR improvement with shuffling and bzip2/gzip.

Fig. 1: Illustration of bzip2 level 1 CR improvement, extra time of encoding and decoding for float32

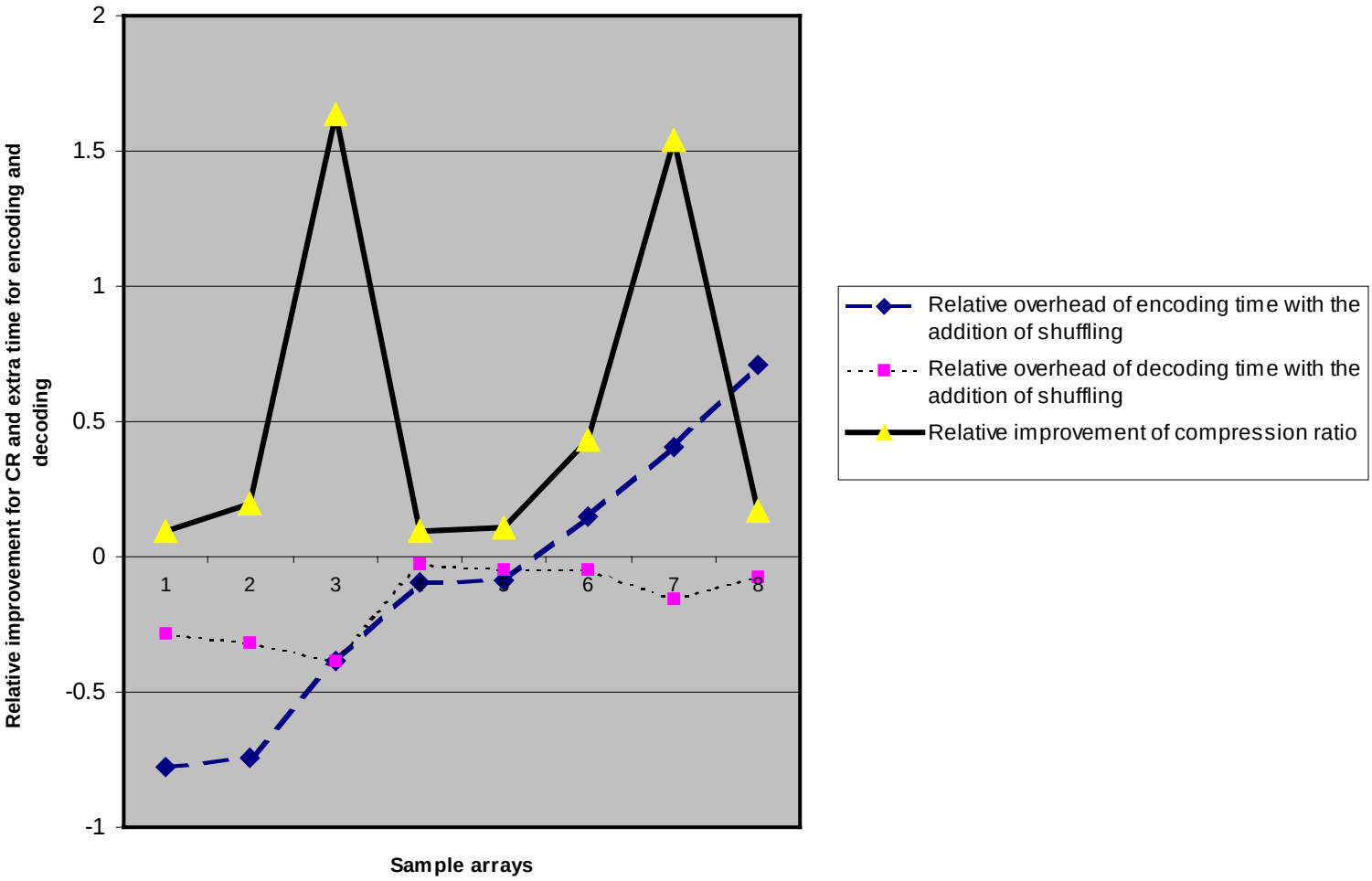


Fig.2: Illustration of bzip2 level 6 CR improvement, extra time of encoding and decoding for float32

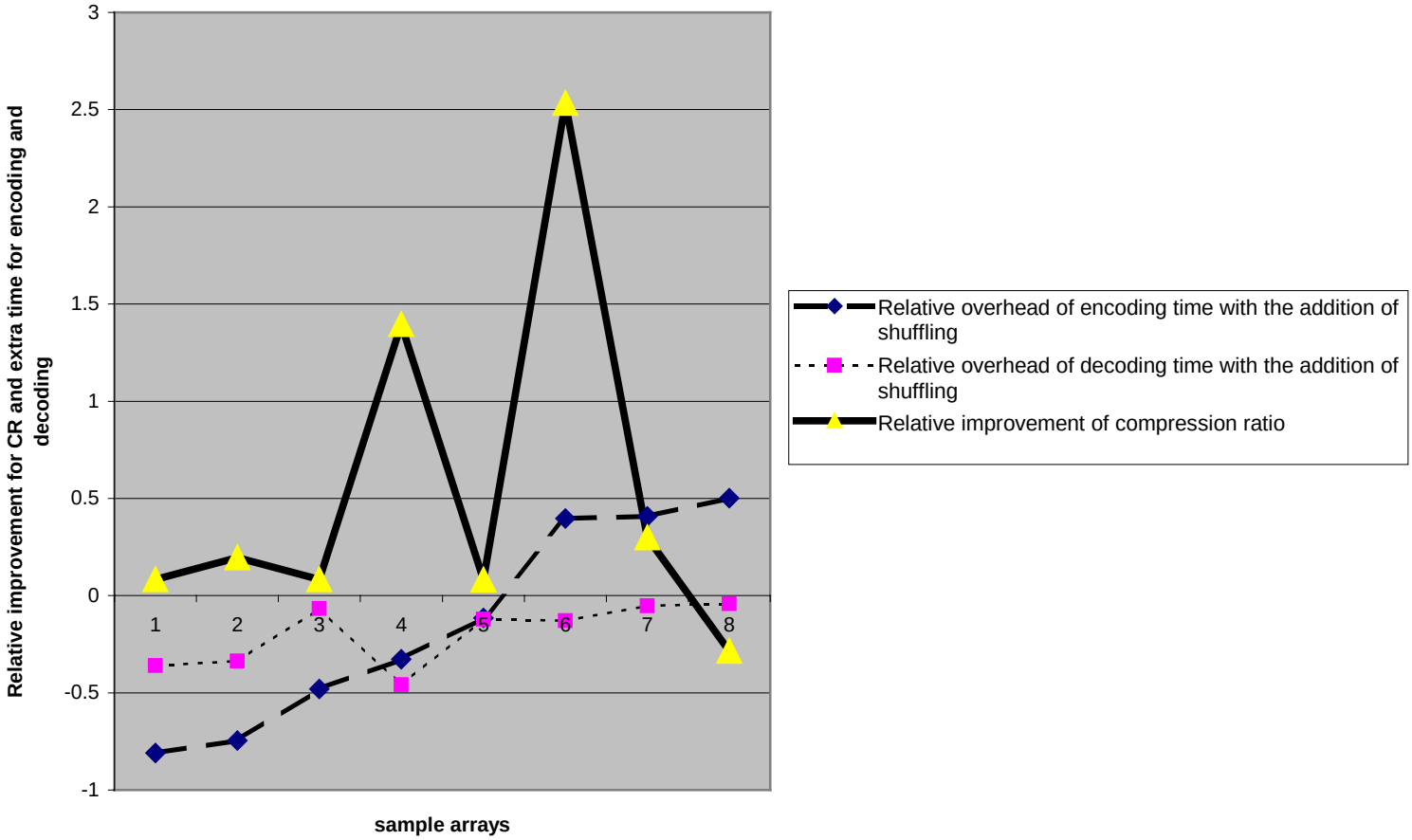


Fig. 3: Illustration of bzip level 9 CR improvement, extra time of encoding and decoding for float32

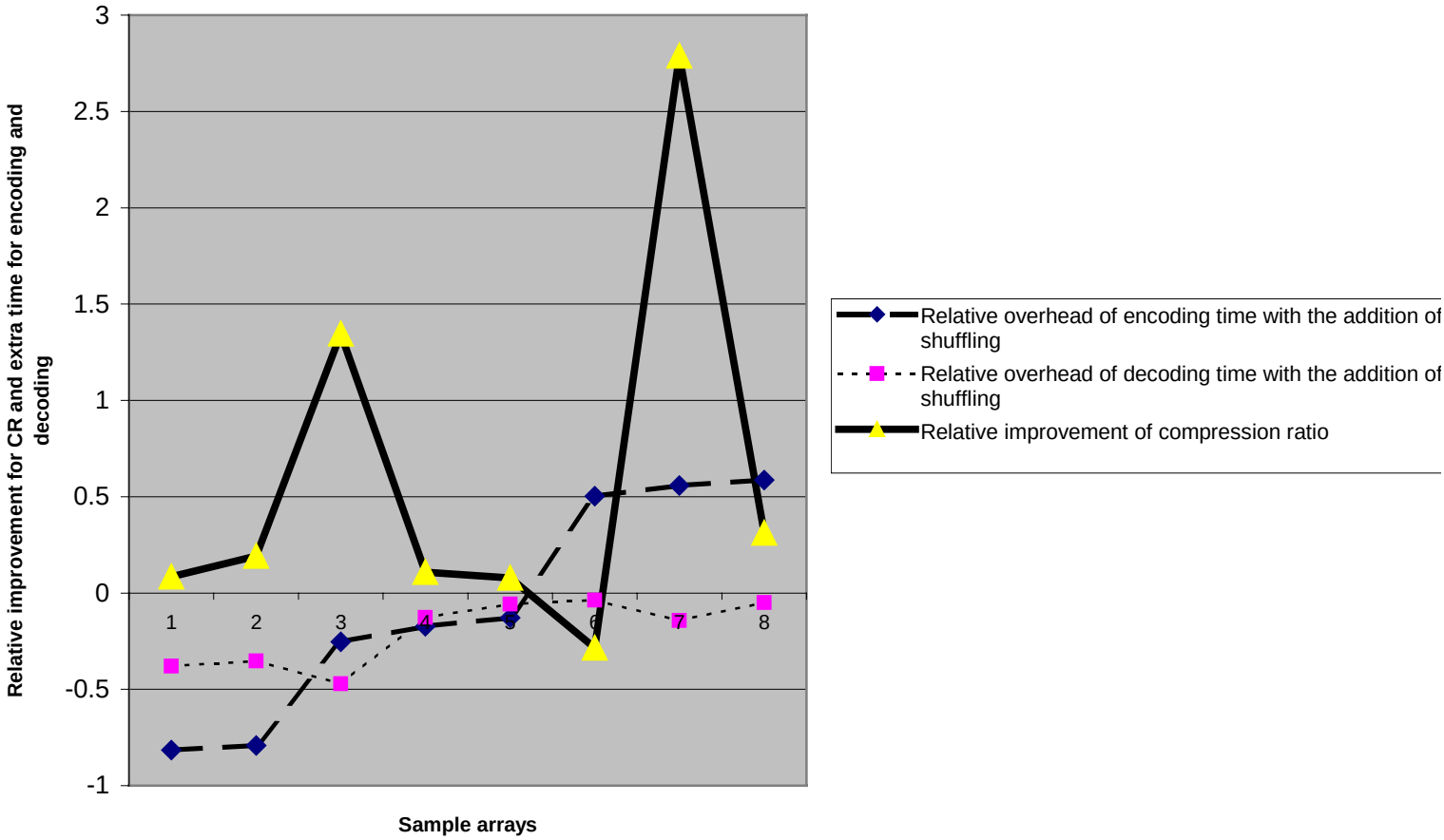


Fig.4 Illustration of CR improvement, extra time of encoding and decoding for float32 data with gzip level 1 compression

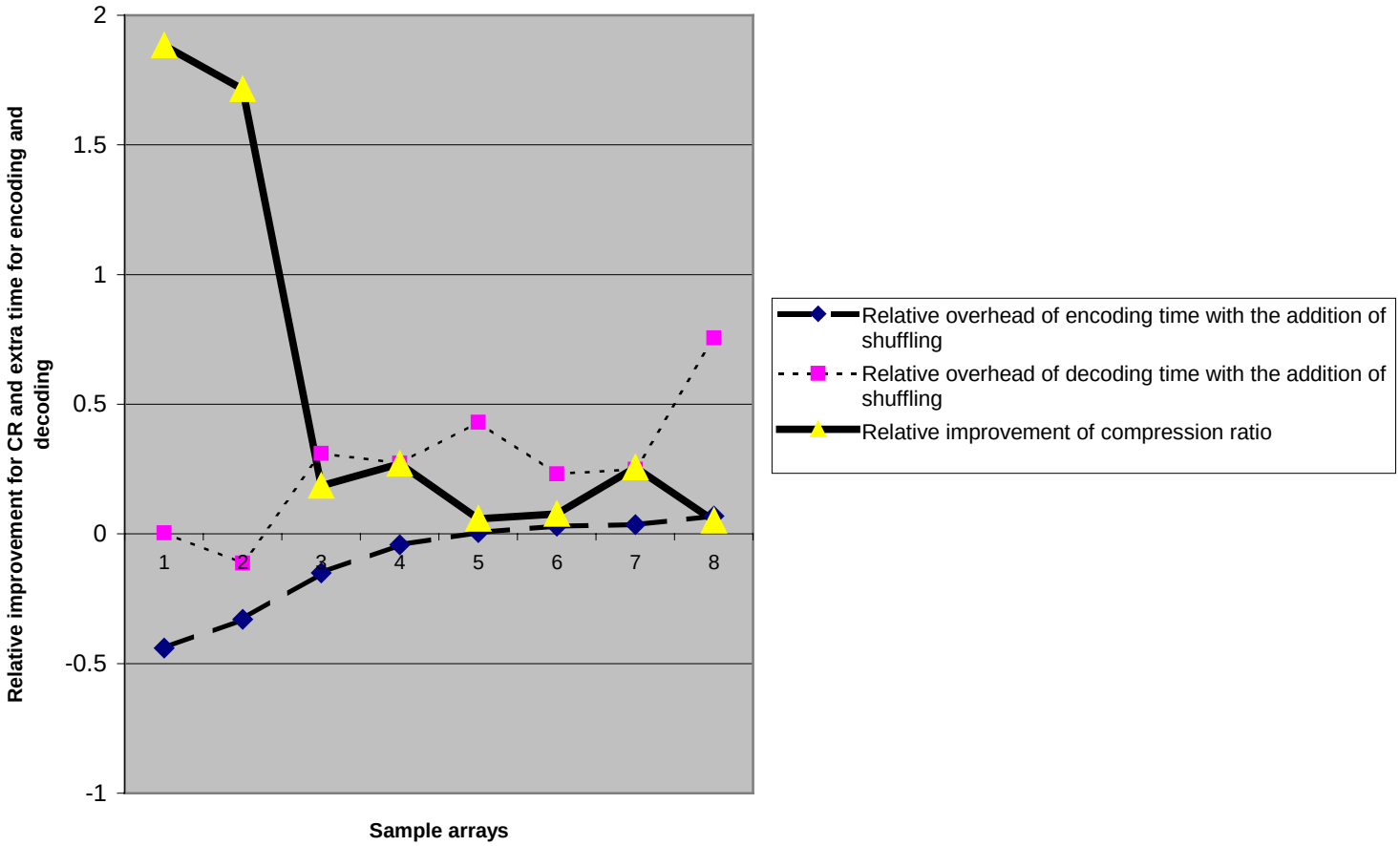


Fig.5: Illustration of CR improvement, extra time of encoding and decoding for float32 data with gzip level 6 compression

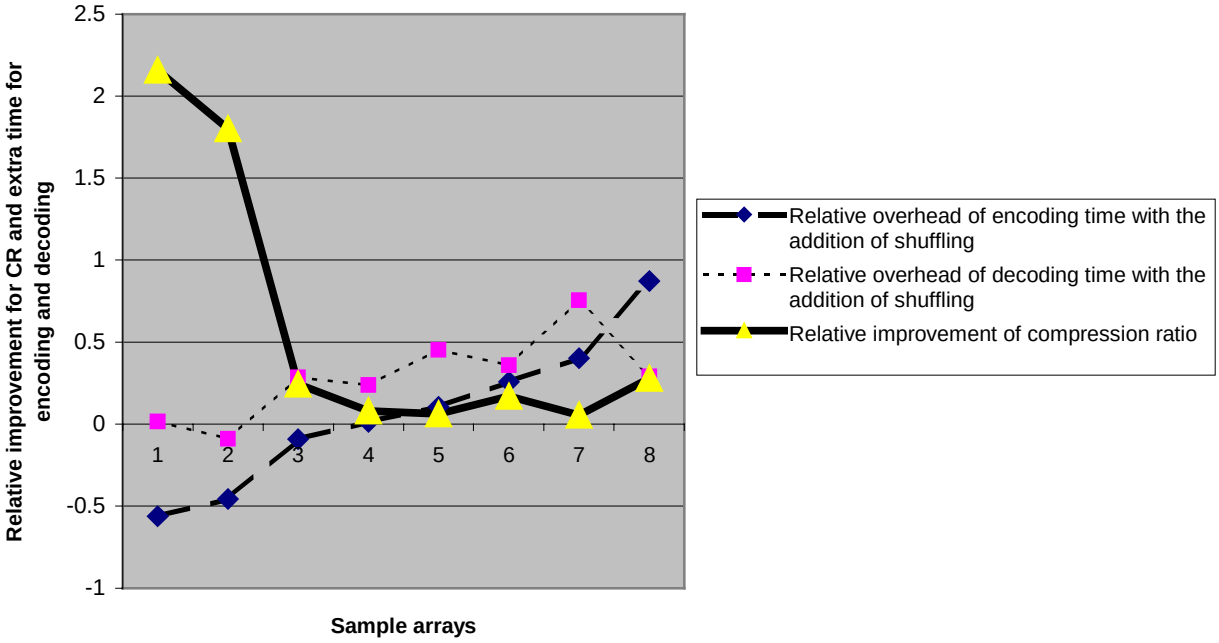


Fig.6: Illustration of CR improvement, extra time of encoding and decoding for float32 data with gzip level 9 compression

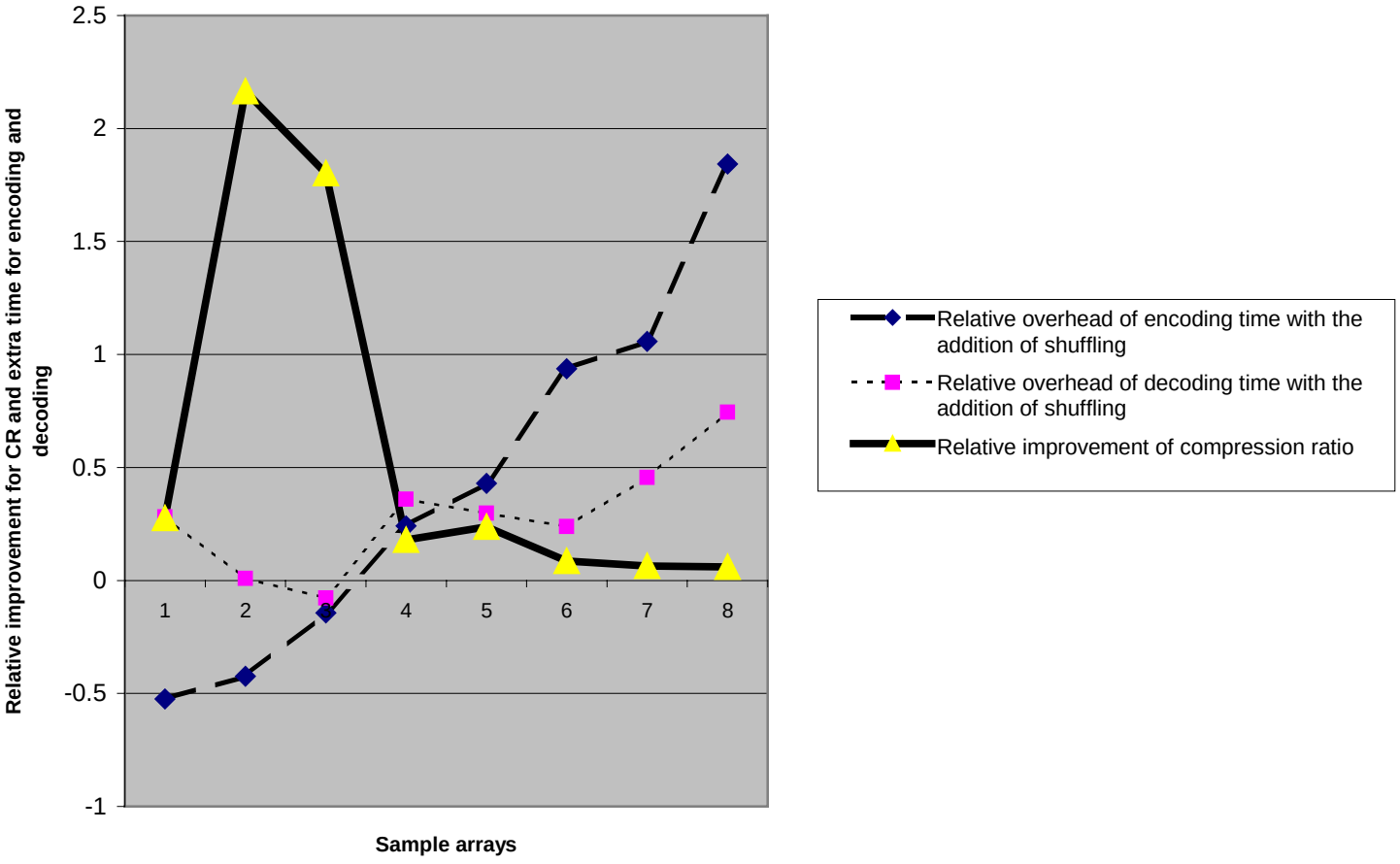


Fig.7: Illustration of CR improvement, extra time of encoding and decoding for float64 data with bzip level 1 compression

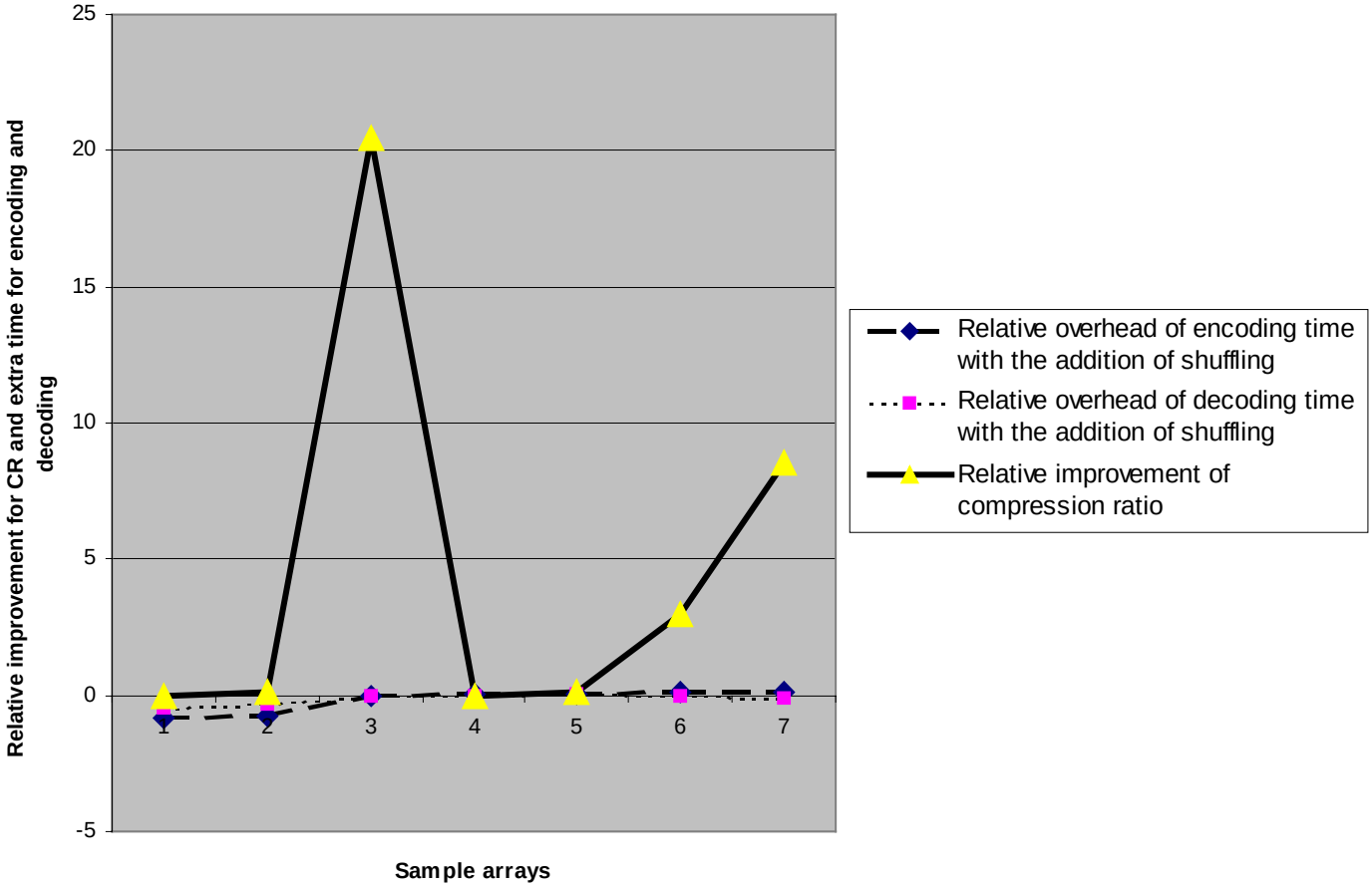


Fig.8: Illustration of CR improvement, extra time of encoding and decoding for float64 data with bzip2 level 6 compression

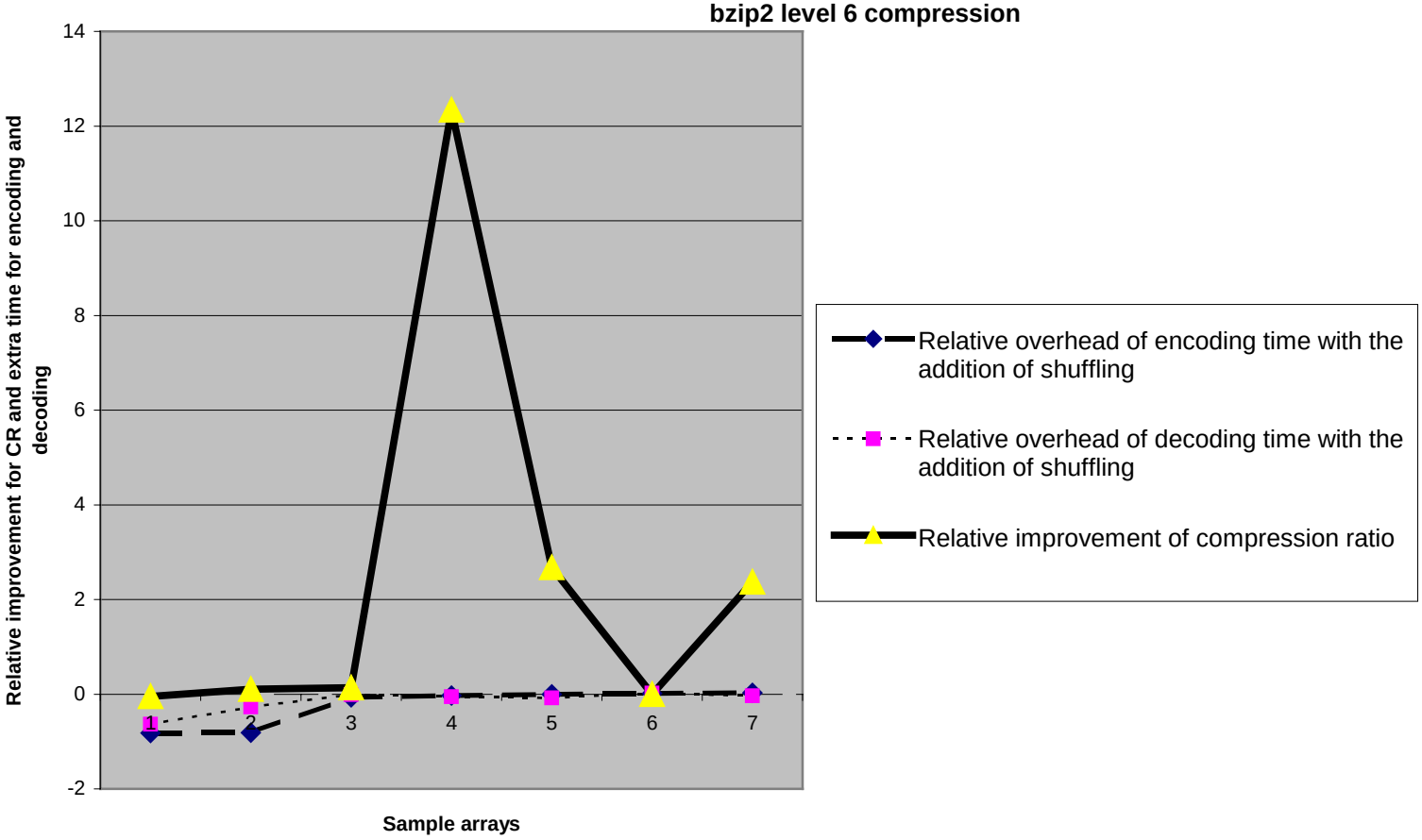


Fig.9: Illustration of CR improvement, extra time of encoding and decoding for float64 data with bzip2 level 9 compression

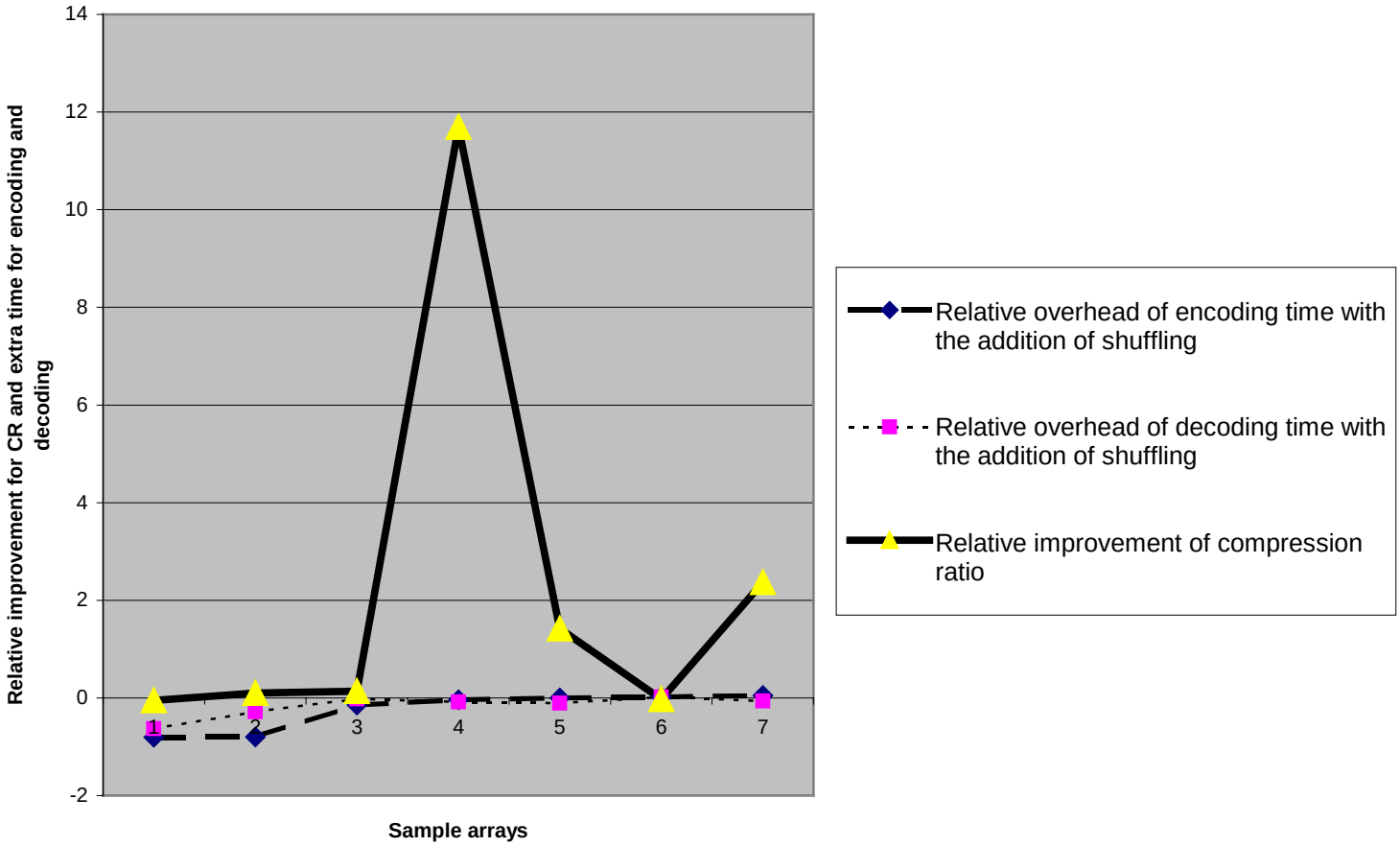


Fig. 10: Illustration of CR improvement, extra time of encoding and decoding for float64 data with gzip level 1 compression

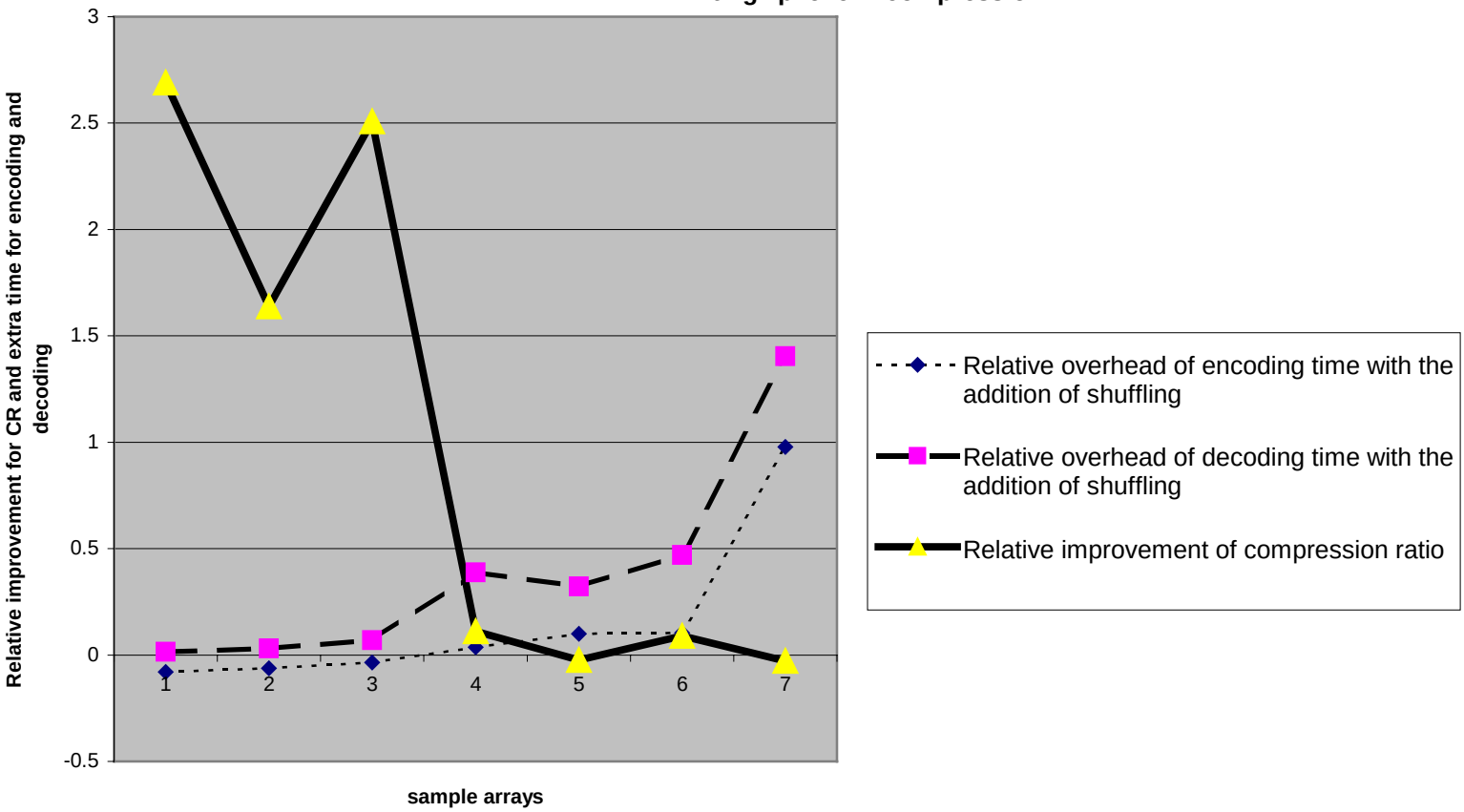


Fig.11: Illustration of CR improvement, extra time of encoding and decoding for float64 data with gzip level 6 compression

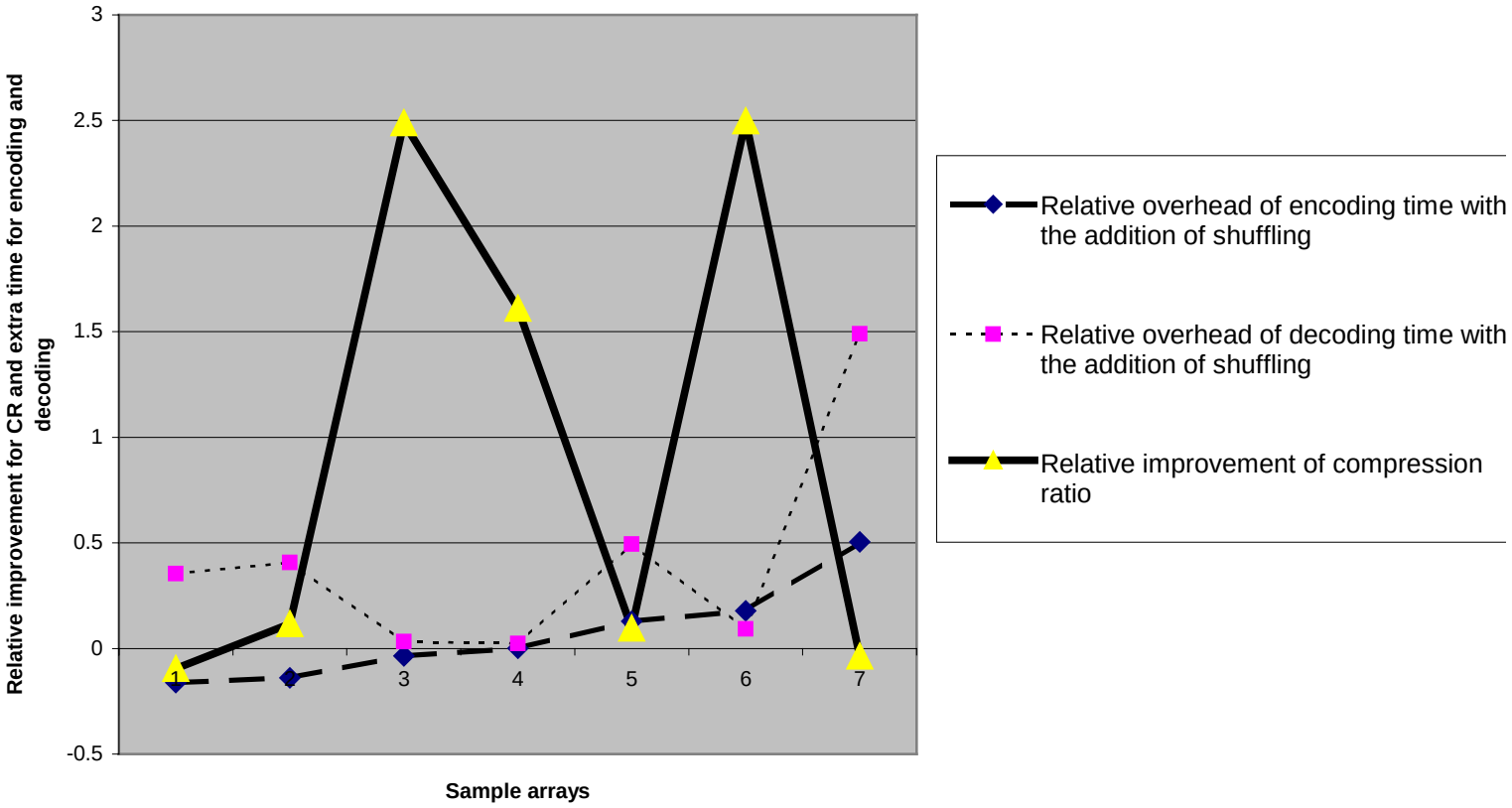
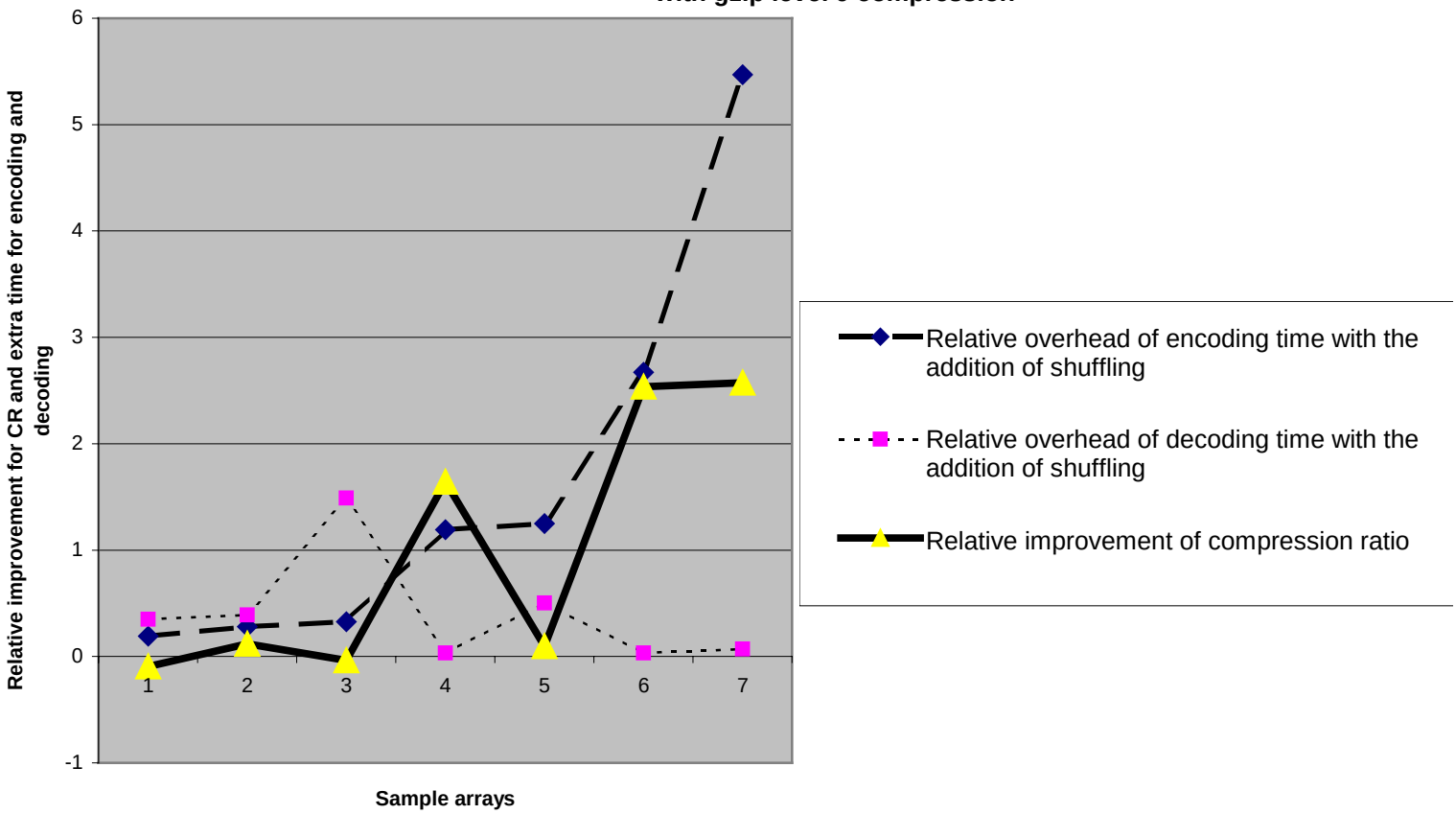


Fig.12: Illustration of CR improvement, extra time of encoding and decoding for float64 data with gzip level 9 compression



8. Concluding remarks

- Integrating shuffling filter to HDF5 library is not hard.
- For floating point scientific data, using the shuffling algorithm with bzip2 or gzip can significantly improve compression ratio.
- It takes less processing time to use shuffling algorithm and bzip2 for data compression and data decompression than to solely use bzip2 only.
- It takes insignificantly extra processing time to use shuffling algorithm and gzip for data compression and data decompression than to solely use gzip only.

9. Future work

Special shuffling algorithm should apply to float32 and float64 data to gain more compression ratio and cut down decoding and encoding time.