Using LibPressio

Robert Underwood, robertu@g.clemson.edu

January 17, 2020

Clemson University

Overview of LibPressio

Problems to Solve

- Every compression library has its own API:
    - More to learn and get correct
    - Proliferation of libraries and tools
    - Little cross pollination

What is LibPressio?

- A generic abstraction for lossy and lossless compression of dense tensors and measurement thereof
  - Simple and Consistent
  - One API for libraries and tools
  - Abstraction for collaboration

What does it support?

- lossy compression: imagemagick (JPEG, WEBP, PNG, GIF, etc...)
- lossless compression: blosc(gzip, lz4, etc...), fpzip
- error bounded compression: sz, zfp, mgard
- measurement: bit rate, compression ratio, compression bandwidth, external scripts, etc...

How simple is it?

It fits on one slide

```c
#include <libpressio.h>
#include <libpressio_ext/io/posix.h>
#include <libpressio_ext/compressors/sz.h>

int main(int argc, char *argv[])
{
  //get the compressor
  struct pressio* library = pressio_instance();
  struct pressio_compressor* sz =
 ↪   pressio_get_compressor(library, "sz");

  //configure, validate, and assign the options
  struct pressio_options* config =
 ↪   pressio_compressor_get_options(sz);
  pressio_options_set_integer(config, "sz:error_bound_mode",
 ↪   REL);
  pressio_options_set_double(config, "sz:rel_err_bound",
 ↪   0.01);
  pressio_compressor_set_options(sz, config);

  //read in an input buffer
  size_t dims[] = {500,500,100};
  struct pressio_data* description =
 ↪   pressio_data_new_empty(pressio_float_dtype, 3, dims);
  struct pressio_data* input_data =
 ↪   pressio_io_data_path_read(description,
 ↪   "CLOUDf48.bin.f32");

  //create output buffers
  struct pressio_data* compressed_data =
 ↪   pressio_data_new_empty(pressio_byte_dtype, 0, NULL);
  struct pressio_data* decompressed_data =
 ↪   pressio_data_new_owning(pressio_float_dtype, 3, dims);

  //compress and decompress the data
  pressio_compressor_compress(sz, input_data,
 ↪   compressed_data);
  pressio_compressor_decompress(sz, compressed_data,
 ↪   decompressed_data);

  //free memory
  pressio_data_free(decompressed_data);
  pressio_data_free(compressed_data);
  pressio_data_free(input_data);
  pressio_options_free(config);
  pressio_compressor_release(sz);
  pressio_release(library);
  return 0;
}
```

What have I done with it?

- Generic CLI for different compressors
- Python and Julia bindings
- An auto-tuning framework (v2 in progress)
- A distributed benchmarking framework (in-progress)

Tutorial

Installing Libpressio

Either:

- Install dependencies and use CMake (See README.md)
- Easy install via Docker:

```
git clone https://github.com/codarcode/libpressio
cd libpressio
docker build -t pressio -f docker/Dockerfile-Fedora "."
docker run -it --rm -v $HOME/data:/data pressio
```

Goal

- Goal: write a program that compresses using SZ in 10 minutes or less
    - Code that is actually useful
    - And learn LibPressio along the way

Overview of Library

- There are 5 major structures in LibPressio
  - pressio – get references to compressors
  - pressio_options – represent a set of options
  - pressio_data – represent data
  - pressio_compressor – compress/decompress
  - pressio_metrics – tooling interface

Example Overview

```c
#include <libpressio.h>
#include <libpressio_ext/io/posix.h>
#include <libpressio_ext/compressors/sz.h>

int main(int argc, char *argv[])
{
  //get the compressor
  struct pressio* library = pressio_instance();
  struct pressio_compressor* sz =
 ↪   pressio_get_compressor(library, "sz");

  //configure, validate, and assign the options
  struct pressio_options* config =
 ↪   pressio_compressor_get_options(sz);
  pressio_options_set_integer(config, "sz:error_bound_mode",
 ↪   REL);
  pressio_options_set_double(config, "sz:rel_err_bound",
 ↪   0.01);
  pressio_compressor_set_options(sz, config);

  //read in an input buffer
  size_t dims[] = {500,500,100};
  struct pressio_data* description =
 ↪   pressio_data_new_empty(pressio_float_dtype, 3, dims);
  struct pressio_data* input_data =
 ↪   pressio_io_data_path_read(description,
 ↪   "CLOUDf48.bin.f32");

  //create output buffers
  struct pressio_data* compressed_data =
 ↪   pressio_data_new_empty(pressio_byte_dtype, 0, NULL);
  struct pressio_data* decompressed_data =
 ↪   pressio_data_new_owning(pressio_float_dtype, 3, dims);

  //compress and decompress the data
  pressio_compressor_compress(sz, input_data,
 ↪   compressed_data);
  pressio_compressor_decompress(sz, compressed_data,
 ↪   decompressed_data);

  //free memory
  pressio_data_free(decompressed_data);
  pressio_data_free(compressed_data);
  pressio_data_free(input_data);
  pressio_options_free(config);
  pressio_compressor_release(sz);
  pressio_release(library);
  return 0;
}
```

Include Required Headers

```c
#include <libpressio.h>
#include <libpressio_ext/io/posix.h>
#include <libpressio_ext/compressors/sz.h>
```

- `libpressio.h` – convenience header for basic usage
- `libpressio_ext/io/posix.h` – POSIX io methods
- `libpressio_ext/compressors/sz.h` – definitions for SZ

struct pressio

- Query:
  - supported compressors
  - version information
- Get/Release instances of compressors
- Error Handling

```
//get the compressor
struct pressio* library = pressio_instance();
struct pressio_compressor* sz =
↪   pressio_get_compressor(library, "sz");
```

struct pressio_options

- Options are runtime settings
- configuration is compile time settings
- Introspect option:
  - names
  - types
- Get/Set/Cast Option values
- Validate options

```
//configure, validate, and assign the options
struct pressio_options* config =
↪  pressio_compressor_get_options(sz);
pressio_options_set_integer(config,
↪  "sz:error_bound_mode", REL);
pressio_options_set_double(config,
↪  "sz:rel_err_bound", 0.01);
pressio_compressor_set_options(sz, config);
```

struct pressio_data

- A generic reference to data
- Helper IO functions
- Query:
  - type
  - size
  - values
  - owning / non-owning
- Extensible

```c
size_t dims[] = {500,500,100};
struct pressio_data* description =
↪  pressio_data_new_empty(pressio_float_dtype,
↪  3, dims);
struct pressio_data* input_data =
↪  pressio_io_data_path_read(description,
↪  "CLOUDf48.bin.f32");

//create output buffers
struct pressio_data* compressed_data =
↪  pressio_data_new_empty(pressio_byte_dtype,
↪  0, NULL);
struct pressio_data* decompressed_data =
↪  pressio_data_new_owning(pressio_float_dtype,
↪  3, dims);
```

struct pressio_compressor

- Compress
- Decompress
- Version info
- Error handling
- Tooling interface (metrics)

```
//compress and decompress the data
pressio_compressor_compress(sz, input_data,
↪  compressed_data);
pressio_compressor_decompress(sz,
↪  compressed_data, decompressed_data);
```

Cleanup

- Well-defined memory model
- "Move"-semantics where possible
- Release v.s. Free

```
//free memory
pressio_data_free(decompressed_data);
pressio_data_free(compressed_data);
pressio_data_free(input_data);
pressio_options_free(config);
pressio_compressor_release(sz);
pressio_release(library);
```

struct pressio_metrics

- Tooling interface
- Every function hooked
- Order Matters!
- Built ins for:
    - size
    - time
    - error statistics
    - error evaluation shell scripts

```
const char* metric_ids[] = {"size", "time"};
struct pressio_metrics* metrics =
↪  pressio_new_metrics(library, metric_ids,
↪  2);
pressio_compressor_set_metrics(sz, metrics);

//use the API

double compression_ratio;
struct pressio_options* results =
↪  pressio_compressor_get_metrics_results(sz);
pressio_options_get_double(results,
↪  "size:compression_ratio",
↪  &compression_ratio);
printf("cr=%lf\n", compression_ratio);
pressio_metrics_free(metrics);
```

What else is there?

- C++ interface for extensions
- Custom compressors
- Custom metrics
- HDF5 file support

What else is there?

- C++ interface for extensions
- Custom compressors
- Custom metrics
- HDF5 file support
- Whatever you all come up with!

Questions?

Using LibPressio

Robert Underwood, robertu@g.clemson.edu

January 17, 2020

Clemson University