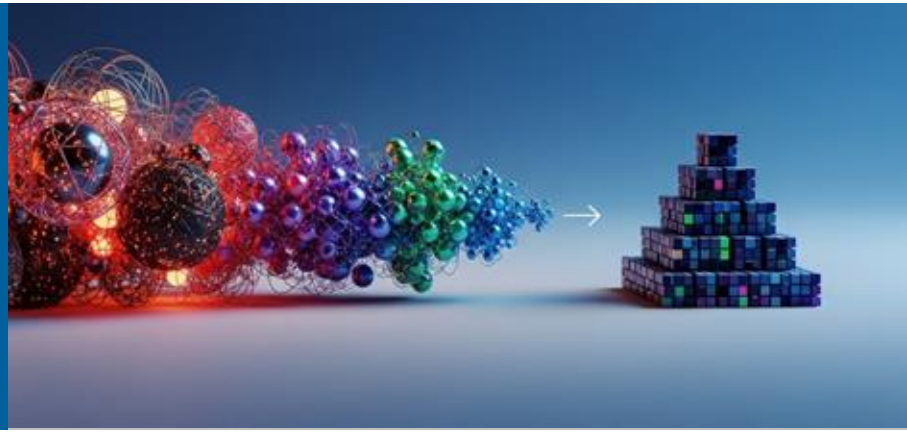
Robert Underwood, Sheng Di, Franck Cappello and Many more from the FZ team

Argonne NATIONAL LABORATORY

LibPressio

Productive and Performant Compression



Robert Underwood
Assistant Computer Scientist
Argonne National Laboratory

Scientist at Large, CASE
University of Chicago

Image Credit: Google Nano Bannana

U.S. DEPARTMENT OF ENERGY Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

LibPressio is an abstraction for compresors

- **Not a compressor, but an interface**
  - Standards for: common bounds, threading, metrics, data specification, compression ratio estimation, max output size queries, and more
  - These standards enable standardized tooling
- **Batteries Included:**
  - **Bindings**: Bash, C, C++, Julia, Python, R, Rust*, WASM*
  - **Libraries:** HDF5, Adios{1,2}, NumCodecs, …
  - **Hardware Support:** CPU (multithreading), GPU, Cerebras†
  - **Vast Support:** 56+ compressors, 18+ io formats, 36 metrics, and more
  - **Documentation:** tutorials, starter code, docs for compressor options

* contributed by Juniper Tryee

† In a future release

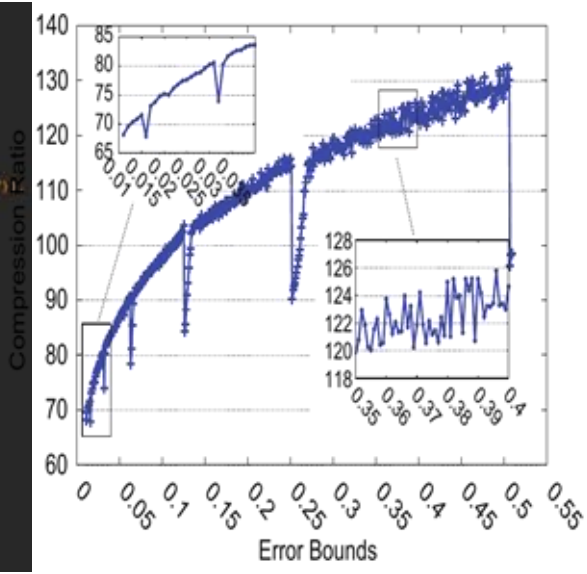
LibPressio enables MetaCompressors

Examples	Function
Pressio	Convert between standard bounds
Chunking	Parallelized and out-of-core compression
LibPressio-Opt	Optimize compressor configurations
LibPressio-ErrorInjector	Sensitivity studies for applications
LibPressio-JIT (aka FZ)	Compile compressors on the fly
Manifest	Encoding provenance into compression
And many more	Common data transformations

Pitch 3: LibPressio-OPT

Sometimes Metrics Look Like This 👇

```
pressio_compressor comp = library.get_compressor("opt");
comp->set_name("pressio");
comp->set_options({
       {"opt:compressor", "many_independent_threaded"},
       {"opt:search", "fraz"},
       {"opt:lower_bound", pressio_data{1e-6, 1e-6}},
       {"opt:upper_bound", pressio_data{1e-1, 1e-1}},
       {"opt:max_iterations", 100u},
       {"opt:objective_mode_name", "max"s},
       {"opt:inputs", std::vector<std::string>{"/pressio/many_independent_threaded/u:pressio:abs", "/pressio/many_independent_threaded/v:pressio:abs"}},
       {"opt:output", std::vector<std::string>{"composite:objective", "size:compression_ratio", "momentum:momentum_error"}},
       {"/pressio/many_independent_threaded:many_independent_threaded:metric", "composite"},
       {"/pressio/many_independent_threaded:many_independent_threaded:compressors", std::vector<std::string>{"sz3", "sz3"}},
       {"/pressio/many_independent_threaded:many_independent_threaded:names", std::vector<std::string>{"u", "v"}},
       {"/pressio:opt:metric", "composite"},
       {"composite:plugins", std::vector<std::string>{"size"s, "time"s, "error_stat"s}},
       {"composite:scripts", std::vector<std::string>{R"(
           local cr = metrics['size:compression_ratio'];
           local percent = metrics['momentum:momentum_error'];
           local objective = 0;
           if cr ~= nil and percent ~= nil then
               if percent <= 1 then
                   objective = cr;
               else
                   objective = 1 - percent;
               end
           end
           return "objective", objective;
       )"s}},
});
```



☝Just ask for what you need

Appendix LibPressio

Installing LibPressio

There are four recommended ways to install LibPressio

- **Spack** develop and use our cutting edge versions
  - `spack repo add https://github.com/robertu94/spack_packages`
  - `spack install libpressio`
- **Conda** stable releases of python packages maintained by SLAC
  - `conda install lcls-ii::libpressio-cpu`
- **Docker** deploy compression as a service, or a tutorial
  - `podman run -it –rm ghcr.io/robertu94/libpressio:latest`
  - `podman run -it –rm ghcr.io/robertu94/libpressio_tutorial:$ARCH`
- **numcodecs-rs** for WASM integration maintained by Juniper Tyree
  - `pip install numcodecs-wasm-pressio`

You can also build from source using cmake. See the README for details

How to compress data with non-finite values

```
pressio_data mask = pressio_data::owning(pressio_uint8_dtype, in->dimensions());
pressio_data_for_each<void>(*in, mask, [](auto in_begin, auto in_end, auto mask_begin){
        while(in_begin != in_end) {
            *mask_begin = std::isnan(static_cast<double>(*in_begin));
            ++in_begin;
            ++mask_begin;
        }});

//make interpolation supports 1d and 2d for now. recast for 2d
in->reshape(in->normalized_dims(2));
mask.reshape(in->normalized_dims(2));

pressio_compressor comp = library.get_compressor("mask_interpolation");
comp->set_options({
    {"mask_interpolation:compressor", "sz3"},
    {"mask_interpolation:mask_mode", "interp"},
    {"mask_interpolation:mask", mask},
    {"pressio:nthreads", std::thread::hardware_concurrency()},
    {"pressio:abs", 0.1},
    {"pressio:metric", "composite"s},
    {"sz3:metric", "noop"s},
    {"composite:plugins", std::vector{"error_stat"s, "size"s, "time"s}},
});
```

How to compress with ABS

```
pressio_compressor comp = library.get_compressor("pressio");
comp->set_options({
        {"pressio:compressor", "sperr"},
        {"pressio:abs", 1e-4},
        {"pressio:metric", "composite"s},
        {"composite:plugins", std::vector{"error_stat"s, "size"s, "time"s}},
});
```

How to use PW_REL bounds

```
pressio_compressor comp = library.get_compressor("pw_rel");
comp->set_options({
        {"pw_rel:abs_comp", "sperr"},
        {"pressio:pw_rel", 0.25 / log2(10)}, //apply change of base formula
        {"pw_rel:metric", "composite"s},
        {"composite:plugins", std::vector{"error_stat"s, "size"s, "time"s}},
});
```

Limitations

- If the underlying compressor does not support the bound well, LibPressio can only do so much to save you and may have to default back to lossless.