MPI Extensions for Payload Compression

—

A proposal and a prototype

Carsten Clauss
December 17, 2025

Ever had trouble fitting all your MPI payload into a communication channel?

Here is the solution developed in DaREXA-F:

```
MPIX_Register_compressor()
```

A proposal for MPI extensions for payload compression

- Flexible and lightweight, ideal also for **lossy compression**
- Easily **register your own** compression algorithm by the application
- Hide compression overhead with **partitioned communication**
- Control the compression parameters via **MPI Info objects**
- **New:** Get feedback on compression via extended **MPI Status**

Santa Claus images generated by MS Copilot.

The proposal is basically just the addition of a single API function.

…everything else happens transparently via registered callbacks.

MPIX_REGISTER_COMPRESSOR(name, req_init_fn, req_done, req_free_fn, deflate_fn, inflate_fn, deregister_fn, info, extra_state)

IN	name	compressor identifier (string)
IN	req_init_fn	callback function invoked to initialize a payload compressor for a communication request
IN	req_done_fn	callback function invoked to get and set status information when a receive request is complete
IN	req_free_fn	callback function invoked to release resources allocated by the compressor for a request
IN	deflate_fn	callback function invoked to deflate a payload buffer before sending
IN	inflate_fn	callback function invoked to inflate a payload buffer after receiving again
IN	deregister_fn	callback function invoked when all resources are to be released at the end
IN	extra_state	extra state for the functions of the compressor

Recent addition: New callback, invoked after completion (and decompression) on the receiver side for getting and manipulating the returned MPI status by the registered compressor.

Santa Claus images generated by MS Copilot.

The callbacks for compression/decompression are invoked by the MPI library.

…and for this, the control parameters are handed over via MPI Info objects.

```
MPI_Info_set(info_compr, "comresssor", "darexa-f_zfp");
MPI_Info_set(info_compr, "compressor_backend", "CPU");
MPI_Info_set(info_compr, "compressor_datatype", "double");
MPI_Info_set(info_compr, "compressor_fixed_rate", "4");
…
MPI_Psend_init(buffer, partitions, part_size, MPI_DOUBLE,
               dest, tag, MPI_COMM_WORLD, info_compr,
               &request);
…
   MPI_Start(&request); // Start partitioned sending scheme
   …
   #pragma omp parallel for shared(request)
   for (int i=0; i < partitions; i++) MPI_Pready(i, request);
   MPI_Wait(&request, &status);
…
MPI_Request_free(&request);
```

Sending side

The callback function for the data compression (`deflate_fn()`) is called **transparently** by the MPI library before the data is being sent. …and the same happens on the receiving side for decompression.

Santa Claus images generated by MS Copilot.

The callbacks for compression/decompression are invoked by the MPI library.

…and for this, the control parameters are handed over via MPI Info objects.

```
MPI_Info_set(info_compr, "comressor", "darexa-f_zfp");
MPI_Info_set(info_compr, "compressor_backend", "CPU");
MPI_Info_set(info_compr, "compressor_datatype", "double");
MPI_Info_set(info_compr, "compressor_fixed_rate", "4");
…
MPI_Precv_init(buffer, partitions, part_size, MPI_DOUBLE,
                dest, tag, MPI_COMM_WORLD, info_compr,
                &request);
…
   MPI_Start(&request); // Start receiving the partitions
   …
   MPI_Wait(&request, &status);
   MPIX_Status_get_info(&status, &info_status);
   MPI_Info_get_string(info_status, "compressor_quality", …);
…
MPI_Request_free(&request);
```

Receiving side

And here another proposal for an MPI extension: With "*status-attached MPI Info objects*", arbitrary information could be conveyed (as string-coded key/value pairs) with an MPI status. Here, for example, is the "quality" of the data received after decompression.

Santa Claus images generated by MS Copilot.

In total, we have developed three proposals for new MPI extensions.

…and for each, we have written a short document in the style of the MPI standard.

1. **The ability to register compressor algorithms.**

   `MPIX_Register_compressor()`

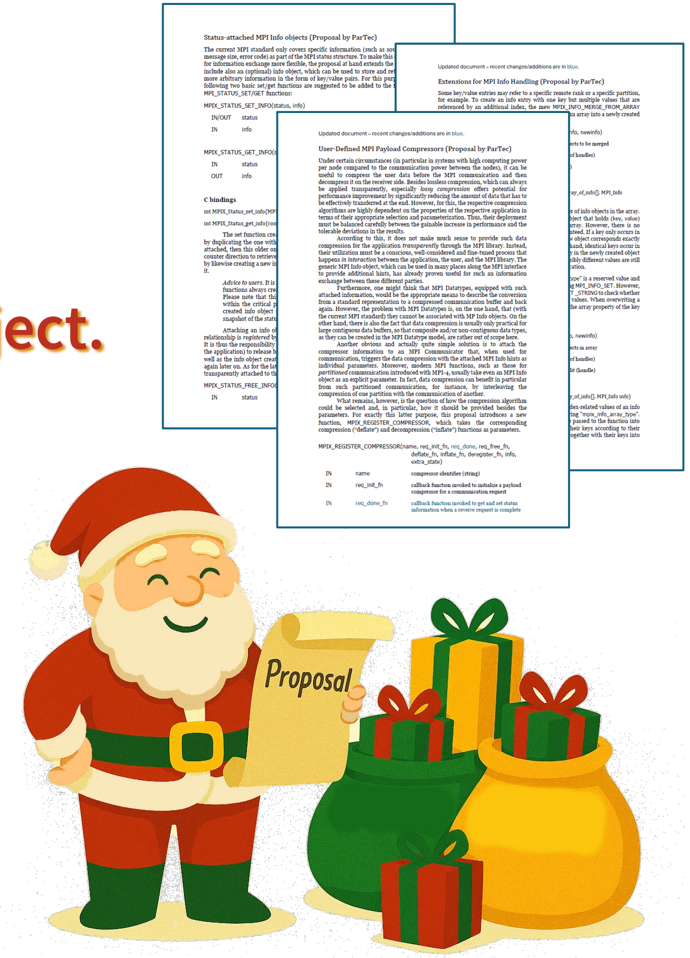
2. **The ability to extend an MPI status with an info object.**

   `MPIX_Status_set/get_info()`

3. **The ability to manage also arrays in info objects.**

   `MPIX_Info_merge_from/split_into_array()`

Plus the ability to store binary info values by MPICH:
`MPIX_Info_set_hex()`



Santa Claus images generated by MS Copilot.

What is also already available is a prototype implementation.

…as we have integrated all extensions into ParaStation MPI.

- ParaStation MPI is a production-grade, high-performance MPI library:
  - Based on MPICH, and developed by ParTec
  - Used, for example, at the Jülich Supercomputing Centre (JSC)
  - Available as open-source on GitHub:
    `https://github.com/ParaStation/psmpi`
- Extensions proposed here are available online:
  - `https://github.com/carsten-clauss/psmpi/tree/cc/darexa-MPIX_Register_compressor`
  - Same as the proposal documentation
    (see **doc** sub-folder in project above)
  - Usage examples and tests are also available:
    `psmpi/mpich2/test/mpi/parastation`

ParaStation MPI

Santa Claus images generated by MS Copilot.

Thank You!