CASS Community BoF Days
June 12, 2024

MPICH: A High-Performance Open Source MPI Library for Leadership-class HPC Systems

Agenda

- Argonne Update – Yanfei Guo
- User presentations
  - Jeff Hammond (NVIDIA)
  - Vitali Morozov (Argonne)
  - Wei-keng Liao (Northwestern University)
  - Jiajun Huang (ANL/University of California, Riverside)
  - Junchao Zhang (ANL)
- Wrap Up/Q&A

MPICH: Status and Upcoming Releases
http://www.mpich.org

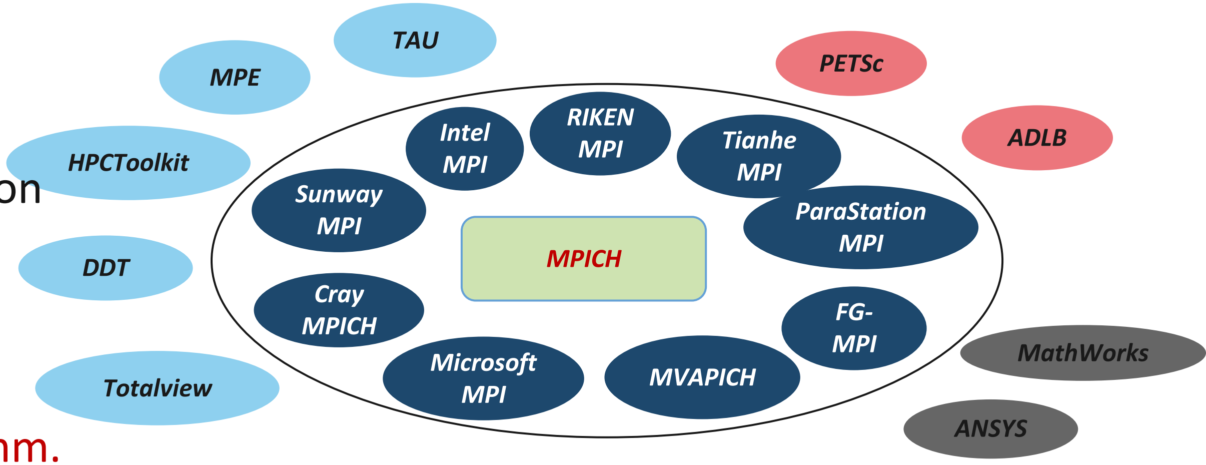
Ken Raffenetti, **Yanfei Guo**, Hui Zhou, Rajeev Thakur

Argonne National Laboratory

MPICH turns 31

The MPICH Project

- Funded by DOE for 31 years
- Has been a key influencer in the adoption of MPI
  - First/most comprehensive implementation of every MPI standard
  - Allows supercomputing centers to not compromise on what features they demand from vendors
- DOE R&D100 award in 2005 for MPICH
- DOE R&D100 award in 2019 for UCX (MPICH internal comm. layer)
- MPICH and its derivatives are the world's most widely used MPI implementations



MPICH is not just a software

It's an Ecosystem

MPICH Adoption in Exascale Machines

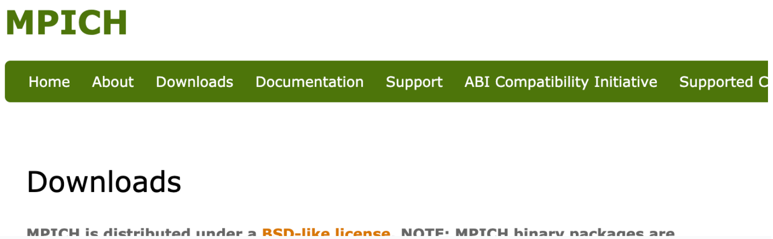
- Aurora, ANL, USA (Intel MPI for Aurora)
- Frontier, ORNL, USA (Cray MPICH)
- El Capitan, LLNL, USA (Cray MPICH)

MPICH ABI Compatibility Initiative

- Binary compatibility for MPI implementations
  - Started in 2013
  - Explicit goal of maintaining ABI compatibility between multiple MPICH derivatives
  - Collaborators:
    - MPICH (since v3.1, 2013)
    - Intel MPI Library (since v5.0, 2014)
    - Cray MPICH (starting v7.0, 2014)
    - MVAPICH2 (starting v2.0, 2017)
    - Parastation MPI (starting v5.1.7-1, 2017)
- Open initiative: other MPI implementations are welcome to join
- http://www.mpich.org/abi
- MPI Standard ABI update in later slides…

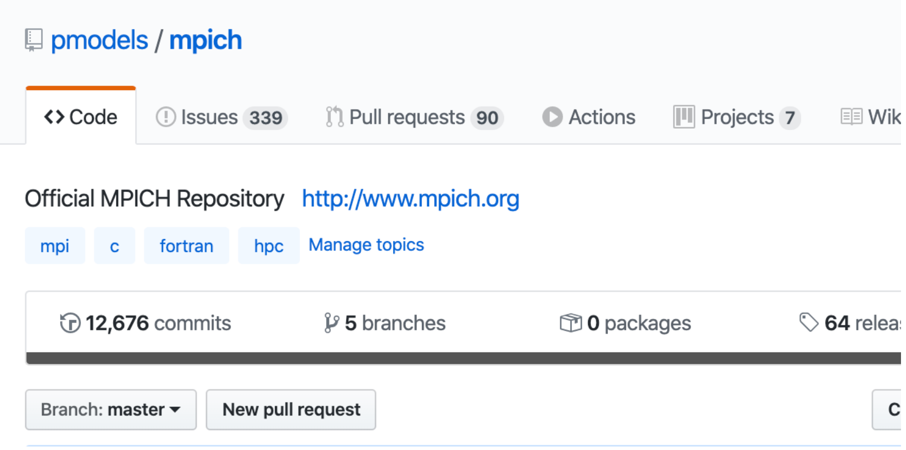
MPICH Distribution Model

- Source Code Distribution
  - MPICH Website, Github
- Binary Distribution through OS Distros and Package Managers
  - Redhat, CentOS, Debian, Ubuntu, Homebrew (Mac)
- Distribution through HPC Package Managers
  - Spack, OpenHPC, E4S
- Distribution through Vendor Derivatives



Downloads

MPICH is distributed under a BSD-like license. NOTE: MPICH binary packages are



Official MPICH Repository http://www.mpich.org

MPICH Support in Spack

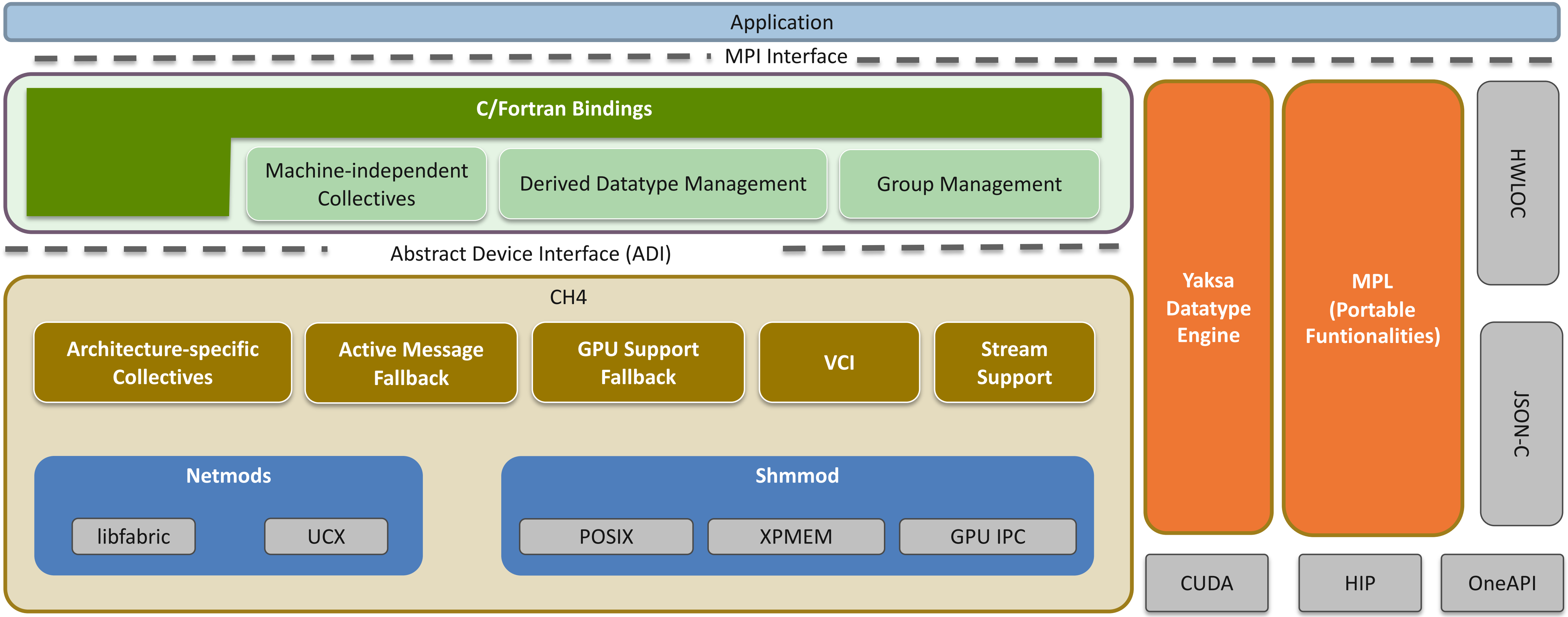
Spack

- Spack package manager is widely used in HPC
- Many MPICH configurations and features supported
- Recently added options
  - XPMEM variant
  - Improved PMI/PMI2/PMIx variants
- We want to hear from you
  - Are there features missing?
  - Are you unable to build/install on your system?
  - Open an issue on Spack Github (https://github.com/spack/spack), use subject “mpich: <…>” and tag @raffenet, @yfguo, @hzhou

MPICH Releases

- MPICH now aims to follow a 12-month cycle for major releases (4.x)
  - Minor bug fix releases for the current stable release happen every few months
  - Preview releases for the next major release happen every few months
  - Branching off when beta is released (feature freezed)
- Current stable release is in the 4.2.x series
  - mpich-4.2.1 released in March, mpich-4.2.2 release by end of June
- Upcoming major release is in the 4.3.x series
  - mpich-4.3.0b1 release targeted for November @ SC24

MPICH Layered Structure

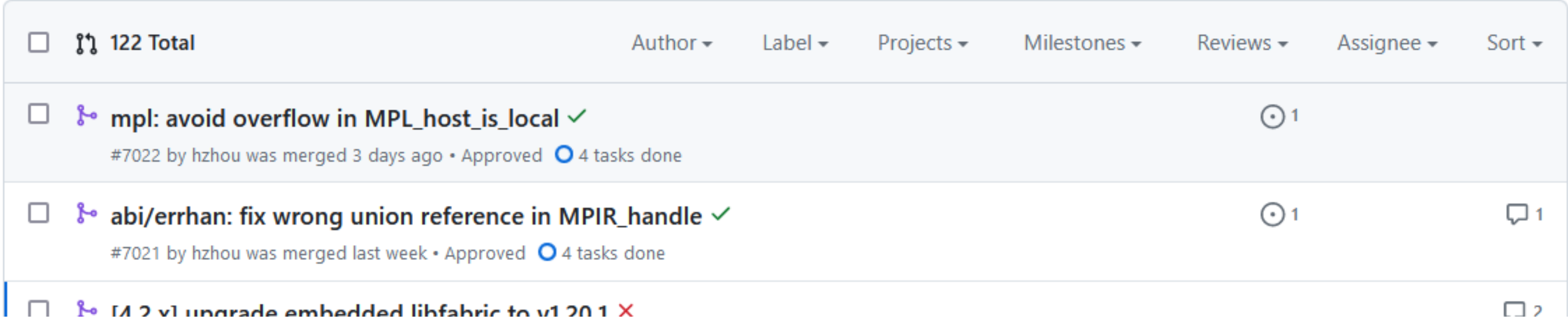


MPICH 4.2

- Full support for MPI 4.1 specification
  - `mpi_memory_alloc_kinds` info hint
  - `MPI_Request_get_status_{all,any,some}`
  - `MPI_Remove_error_{class,code,string}`
  - `MPI_{Comm,Session}_{attach,detach}_buffer`
  - `MPI_BUFFER_AUTOMATIC`
  - Split type `MPI_COMM_TYPE_RESOURCE_GUIDED`
- New experimental features
  - MPI Thread communicator
  - MPI datatype iov query
  - Reduction operator `MPIX_EQUAL`
- Enhanced GPU (esp. ZE) support
- Unified PMI-{1,2,x} support

MPICH 4.3 Update

- Support the new MPI ABI proposal --enable-mpi-abi
- MPIX Async extension – for interoperable MPI progress
  - Custom progress engine can include MPI progress
  - MPI progress can advance custom asynchronous tasks
- Stability and performance issues from Aurora
- Misc fixes and enhancements – 122 merged pull requests so far



☐ 122 Total — Author ▾ Label ▾ Projects ▾ Milestones ▾ Reviews ▾ Assignee ▾ Sort ▾

☐ mpl: avoid overflow in MPL_host_is_local ✓ — ⊙ 1
#7022 by hzhou was merged 3 days ago • Approved ○ 4 tasks done

☐ abi/errhan: fix wrong union reference in MPIR_handle ✓ — ⊙ 1 — 🗨 1
#7021 by hzhou was merged last week • Approved ○ 4 tasks done

☐ [4.2.x] upgrade embedded libfabric to v1.20.1 ✗ — 🗨 2

Support for MPI ABI

- Standardized ABI by MPI Forum
  - Portability across different MPI implementations.
  - Simplify package and dependency management of HPC software
- Try today by building MPICH with --enable-mpi-abi
  - Existing MPICH ABI is offered in parallel
- New compiler wrappers
  - mpicc-abi, mpic++-abi

Jeff R. Hammond, Lisandro Dalcin, Erik Schnetter, Marc Pérache, Jean-Baptiste Besnard, Jed Brown, Gonzalo Brito Gadeschi, Joseph Schuchart, Simon Byrne, and Hui Zhou. MPI Application Binary Interface Standardization. In Proceedings of EuroMPI 2023: the 30th European MPI Users' Group Meeting (EUROMPI '23), September 11–13, 2023, Bristol, United Kingdom. ACM, New York, NY, USA. https://doi.org/10.1145/3615318.3615319

New Extension - MPIX_Op_create_x

- The “old” op user function caters to a Fortran calling convention.

```
typedef void (MPI_User_function)(void *invec, void *inoutvec,
                                 int *len, MPI_Datatype *datatype);
```

- It assumes integer handles, which won’t work with Fortran.
- It won’t work with any non-C/C++ user functions.
- Current MPICH Fortran binding relies on non-standard, language-specific ABIs.

```
void MPII_Op_set_fc(MPI_Op);
void MPII_Op_set_cxx(MPI_Op);
```

- Proposed fix – add a context and a destructor to support binding proxy functions.

```
int MPIX_Op_create_x(MPIX_User_function_x *user_fn_x,
                     MPIX_Destructor_function *destructor_fn,
                      int commute, void *extra_state, MPI_Op *op);
Typedef void (MPIX_User_function_x)(void *invec, void *inoutvec,
                                   MPI_Count len, MPI_Datatype datatype,
                                   void *extra_state);
Typedef void (MPIX_Destructor_function)(void *extra_state);
```

New Extensions to Enable Inter-operable MPI Progress

- Explicit MPI progress
- MPIX Async
- Lightweight request completion query

```
int MPIX_Stream_progress(MPIX_Stream stream);
```

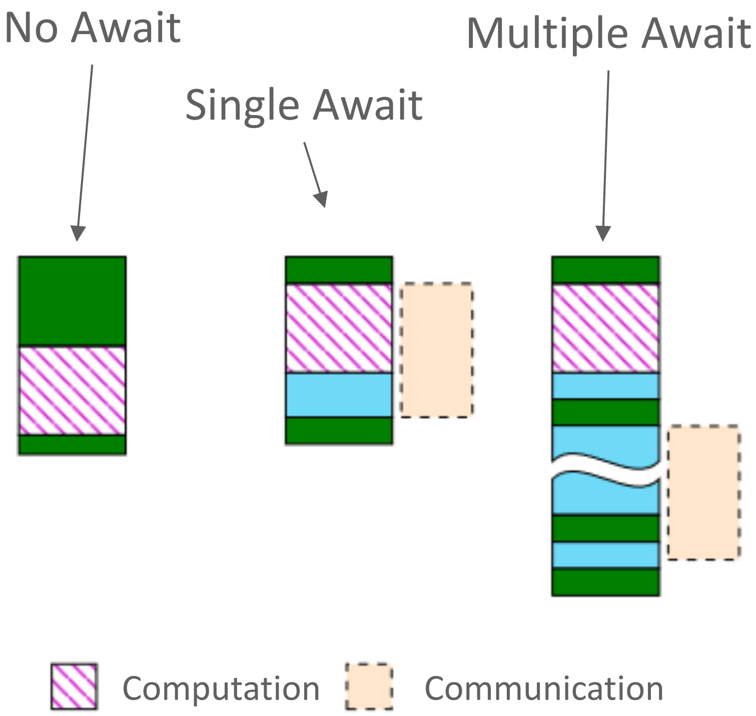
```
int MPIX_Async_start(MPIX_Async_poll_function poll_fn,
                      void *extra_state, MPIX_Stream stream);

enum {
    MPIX_ASYNC_PENDING = 0,
    MPIX_ASYNC_DONE = 1,
};

typedef struct MPIR_Async_thing *MPIX_Async_thing;
typedef int (MPIX_Async_poll_function)(MPIX_Async_thing);

void *MPIX_Async_get_state(MPIX_Async_thing async_thing);

void *MPIX_Async_spawn(MPIX_Async_thing async_thing,
                        MPIX_Async_poll_function poll_fn,
                        void *extra_state, MPIX_Stream stream);
```

```
bool MPIX_Request_is_complete(MPI_Request request);
```

Hui Zhou, Robert Latham, Ken Raffenetti, Yanfei Guo and Rajeev Thakur. MPI Progress For All. https://arxiv.org/pdf/2405.13807

The problem of “fancy” communications

- Three Async Patterns
  - No Await - *e.g.* light weight send
  - Single Await – *e.g.* “strong progress”
  - Multiple Await – *e.g.* fancy schemes require handshakes
- Good computation/communication overlaps are only possible with single await patterns.
- It is more common to require fancy schemes for communication performance due to increasingly hybrid systems.

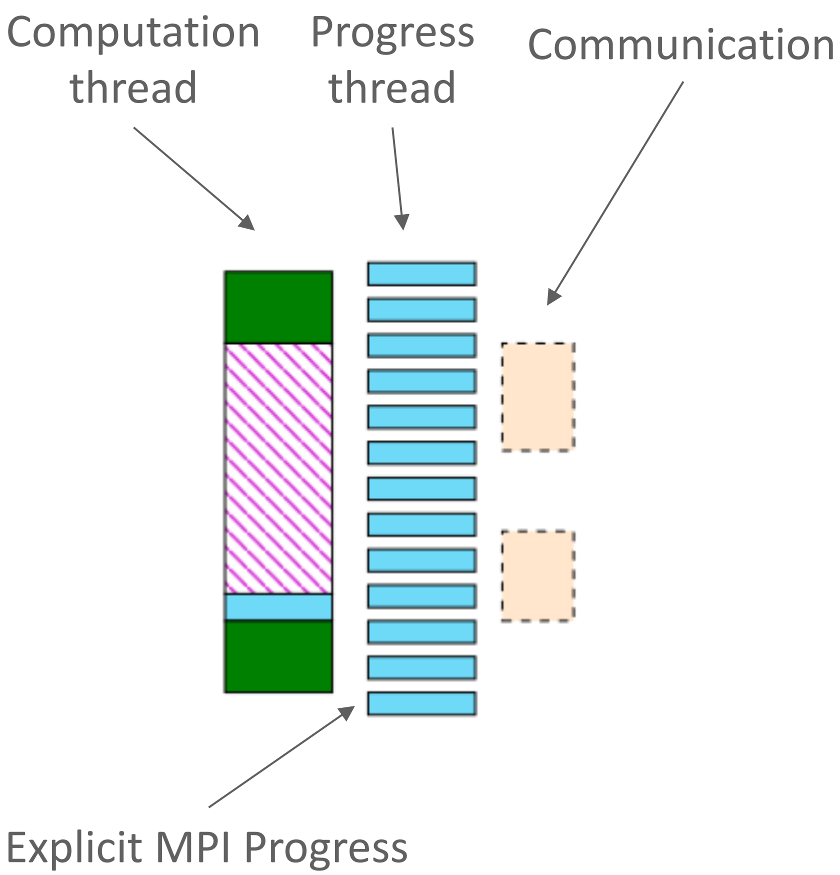


Why we need explicit MPI progress

- To achieve computation/communication overlap, we require a progression scheme, e.g. a progress thread.
- Default global async thread does not work
  - Waste resource when it is not needed
  - Severely degrade performance due to thread contentions
- Solution – explicit MPI progress

```
int MPIX_Stream_progress(MPIX_Stream stream);
```

  - On-demand invocation
  - Per-stream progress



Integrate custom progress hooks into MPI progress

- Enable users to extend MPI by building custom communication algorithms
- Integrate custom progress hooks –
  - Allows for seamless MPI framework, minimize the effort of porting applications
  - Avoid the complexity of building separate progression mechanisms
  - Achieve equivalent performance to a native MPI implementation

```
int MPIX_Async_start(MPIX_Async_poll_function poll_fn,
                     void *extra_state, MPIX_Stream stream);
```

Lightweight request completion query

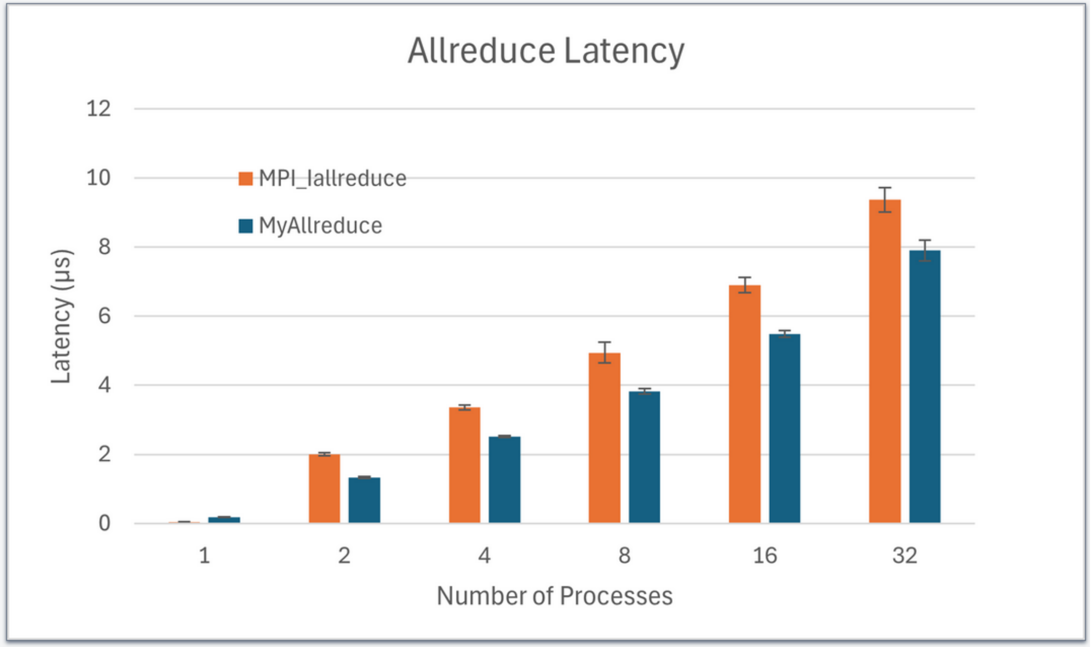
- Asynchronous workflow need to check dependency status
- MPI_Test invokes MPI_Progress
  - It contends with progress engine
  - It does more than what is needed – filling status and freeing requests
- MPIX_Request_is_complete is
  - Lightweight (essentially an atomic query).
  - No side effects.

```
bool MPIX_Request_is_complete(MPI_Request request);
```

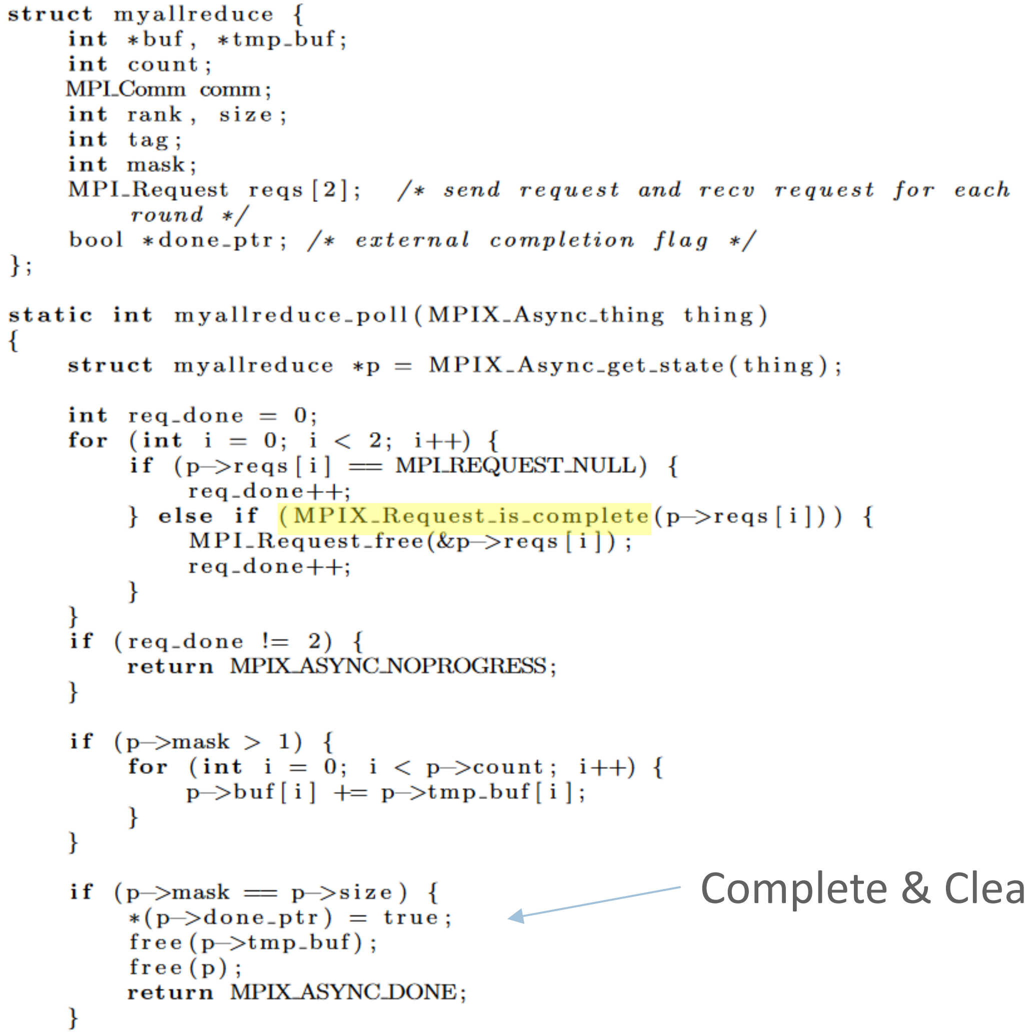
Example: Allreduce Implementation outside of a MPI Library

- Recursive doubling algorithm implemented in outside vs inside an MPI library.

- “MyAllreduce” assumes MPI_IN_PLACE, MPI_INT, MPI_SUM, and a power-of-2 communicator size.
- It out-performs the native implementation due to these assumptions (shortcuts).

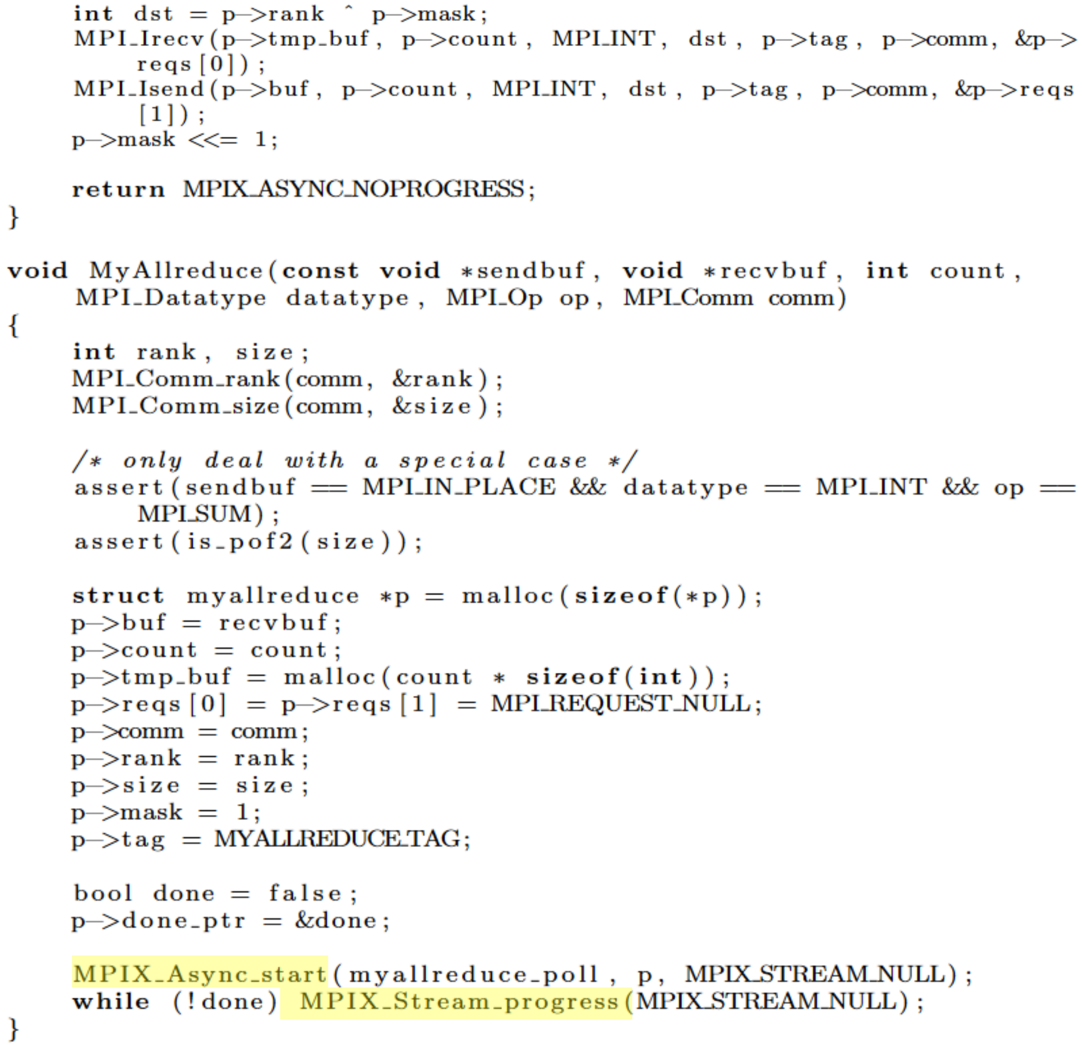


Example: custom Allreduce



```
struct myallreduce {
    int *buf, *tmp_buf;
    int count;
    MPI_Comm comm;
    int rank, size;
    int tag;
    int mask;
    MPI_Request reqs[2];  /* send request and recv request for each
        round */
    bool *done_ptr; /* external completion flag */
};

static int myallreduce_poll(MPIX_Async_thing thing)
{
    struct myallreduce *p = MPIX_Async_get_state(thing);

    int req_done = 0;
    for (int i = 0; i < 2; i++) {
        if (p->reqs[i] == MPI_REQUEST_NULL) {
            req_done++;
        } else if (MPIX_Request_is_complete(p->reqs[i])) {
            MPI_Request_free(&p->reqs[i]);
            req_done++;
        }
    }
    if (req_done != 2) {
        return MPIX_ASYNC_NOPROGRESS;
    }

    if (p->mask > 1) {
        for (int i = 0; i < p->count; i++) {
            p->buf[i] += p->tmp_buf[i];
        }
    }

    if (p->mask == p->size) {
        *(p->done_ptr) = true;
        free(p->tmp_buf);
        free(p);
        return MPIX_ASYNC_DONE;
    }
```

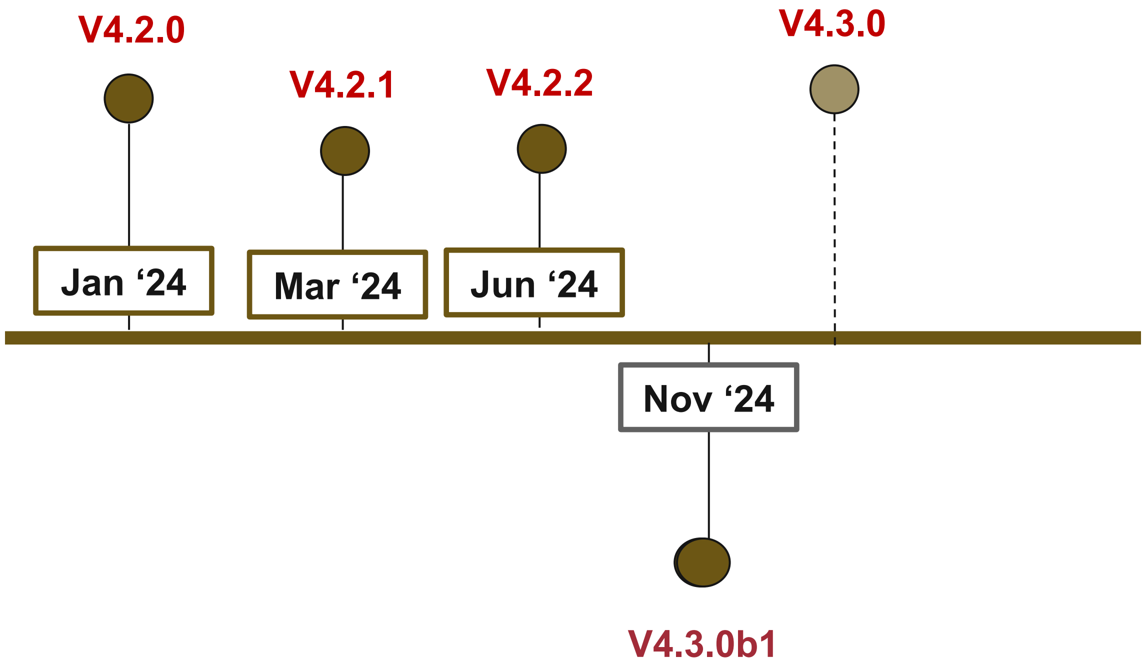
Complete & Cleanup



```
    int dst = p->rank ^ p->mask;
    MPI_Irecv(p->tmp_buf, p->count, MPI_INT, dst, p->tag, p->comm, &p->
        reqs[0]);
    MPI_Isend(p->buf, p->count, MPI_INT, dst, p->tag, p->comm, &p->reqs
        [1]);
    p->mask <<= 1;

    return MPIX_ASYNC_NOPROGRESS;
}

void MyAllreduce(const void *sendbuf, void *recvbuf, int count,
    MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
{
    int rank, size;
    MPI_Comm_rank(comm, &rank);
    MPI_Comm_size(comm, &size);

    /* only deal with a special case */
    assert(sendbuf == MPI_IN_PLACE && datatype == MPI_INT && op ==
        MPI_SUM);
    assert(is_pof2(size));

    struct myallreduce *p = malloc(sizeof(*p));
    p->buf = recvbuf;
    p->count = count;
    p->tmp_buf = malloc(count * sizeof(int));
    p->reqs[0] = p->reqs[1] = MPI_REQUEST_NULL;
    p->comm = comm;
    p->rank = rank;
    p->size = size;
    p->mask = 1;
    p->tag = MYALLREDUCE_TAG;

    bool done = false;
    p->done_ptr = &done;

    MPIX_Async_start(myallreduce_poll, p, MPIX_STREAM_NULL);
    while (!done) MPIX_Stream_progress(MPIX_STREAM_NULL);
}
```

MPICH 4.3.0 Roadmap

- MPICH-4.3.0b1 in November 2024
  - 4.3.x branch is created
- GA release in late 2024/early 2025
- Critical bug fixes are backported to 4.2.x



Thank you!

- https://www.mpich.org
- Mailing list: discuss@mpich.org
- Issues and Pull requests: https://github.com/pmodels/mpich
- Weekly development call every Thursday at 9am (central): https://bit.ly/mpich-dev-call