Lessons from MPICH

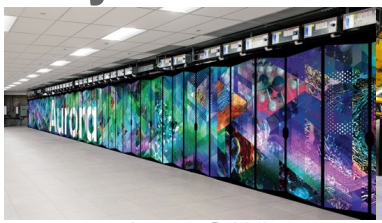
William Gropp
wgropp.cs.Illinois.edu

ACM Software System Award

- Awarded to an institution or individual(s) recognized for developing a software system that has had a lasting influence, reflected in contributions to concepts, in commercial acceptance, or both. The Software System Award carries a prize of $35,000. Financial support for the Software System Award is provided by IBM.
- Previous winning projects include gcc, Unix, TeX, NCSA Mosaic, Java, LLVM, …
  MPICH is the first HPC software project to receive this award
- Citation for ACM: “William Gropp, University of Illinois; Pavan Balaji, Meta; Rajeev Thakur, Yanfei Guo, Kenneth Raffenetti, and Hui Zhou (all of Argonne National Laboratory), receive the ACM Software System Award for MPICH , which has powered 30 years of progress in computational science and engineering by providing scalable, robust, and portable communication software for parallel computers.”

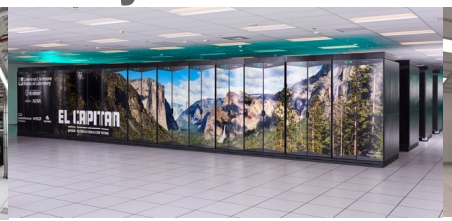
Many people have been involved in MPICH over the years, both working directly on the project and submitting bugs and fixes. All contributed to the success of MPICH and share in this honor.

Major systems worldwide powered by MPICH-based MPI implementations



Aurora@ANL


Frontier@ORNL

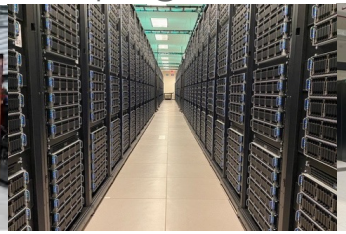


El Capitan@LLNL

Perlmutter@NERSC

Venado@LANL

El Dorado@SNL

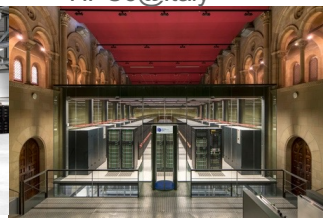

Frontera@TACC

HPC6@Italy

Alps@Swiss

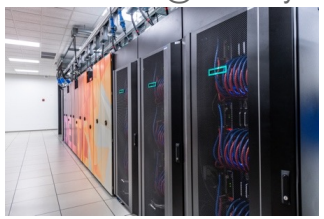
LUMI@Finland

JUWELS@Germany


MareNostrum 4@Spain

Setonix@Australia

Polaris@ANL


DeltaAI@NCSA


Sequoia@LLNL

Birth of MPI and MPICH

- To succeed, the MPI Standard needed implementations
  - Not just any implementation! Required were
    - Performant (low latency, high bandwidth), not just Functional
    - Transportable to a wide variety of systems
    - Open Source so vendors could build on a software base
    - Able to exploit different hardware/system features
      - Remember, no standard networking, no standard CPU architecture, even byte ordering not standard
- MPICH built on Chameleon, a portability layer on top of other message passing systems
  - Chameleon designed as extremely lightweight layer
    - Used for 1992 winner of Gordon Bell Prize for Speedup
    - Direct macro substitution to Intel NX, IBM EUI, TMC CMMD, …
  - MPICH focused on performance, particularly avoiding unnecessary memory copies
- P4 was one of those message passing systems
  - Used to provide MPICH support for communication through shared memory and sockets
  - Provided portability to nearly everything

Birth of MPICH

- Rusty and I committed to “Really run everywhere”
  - There was another popular system that counted “runs on host” as the same as “runs on system”.
  - Both P4 and MPICH had a strong commitment to running on (nearly) everything
  - Remember, the 1990’s was an era of innovation in parallel computing systems
  - The SC Test: How long until a new machine vendor admits that their MPI is MPICH?
    - Rusty and I used to wander the SC show floor and question vendors of parallel systems to see how long it took them to admit their MPI was based on MPICH

Birth of MPICH

- Actively worked with vendors on high-performance ports
  - Understand the capabilities of the hardware
  - Ensure MPI provides access to performance
  - Rusty and I spent a week in St Augustin, Germany, working with NEC on a port to the SX-4
    - Exploit vector architecture and instructions; very high memory bandwidth
    - W. Gropp and E. Lusk. A high-performance MPI implementation on a shared-memory vector supercomputer. Parallel Computing, 22(11):1513–1526, January 1997.
    - At a meeting at LANL, we showed the SX-4 performance – which literally was jaw-dropping for LANL researchers

Goals of MPICH

- Open Source
- Easily customized by others (esp. vendors) for different interconnects, potentially with advanced capabilities
- Performance, especially low latency
  - Low latency is critical for programmability and often performance
- Run everywhere through adherence to standards, OS agnostic
  - Started before Linux existed and when Unix was just one OS choice among several for scientific computing
  - MPICH designed to allow incremental implementation; start with a run-everywhere model then add extra-value features, such as collectives in network
  - Many vendors able to use MPICH as the basis for their own MPI

MPICH Was an Open Source Pioneer

- Early license predates most Open Source licenses (and the term “Open Source” itself)
  - BSD-like (Early BSD and GPL licenses existed. Issues with “copyleft” and derivative software were already understood)
- Worked with ANL legal to draft
  - Goal was to promote adoption, as that had the most value for the US Government, rather than revenue from licenses
- Microsoft adopted MPICH
  - I believe this was the first open source software Microsoft adopted
    - Permissive license was critical

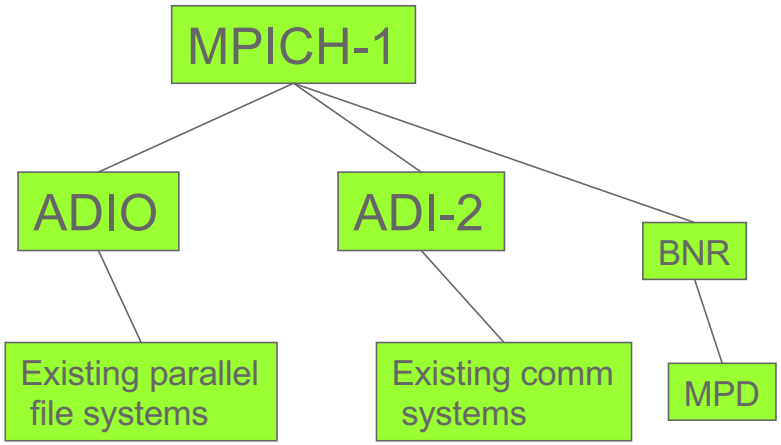
MPICH Design Overview

- Philosophy
  - Performance is essential (make tradeoffs in favor of performance)
  - Must enable vendor innovation in networks, interprocess communication
  - Must be easily portable to all distributed memory parallel systems
- Evolution
  - Early systems had one process per processor, and one processor per node. This shows up in the confusion between processes and processors in early materials about MPI (and other message passing systems)
  - Design assumed lightweight layers, allowing direct use of existing vendor-specific message passing layers
  - Over time, nodes became more complex, with shared memory, accelerators, multiple NICs
- Components
  - Layered design of components, such as collectives and datatypes (and later IO), but allowing custom implementation even at a low level (see philosophy above)

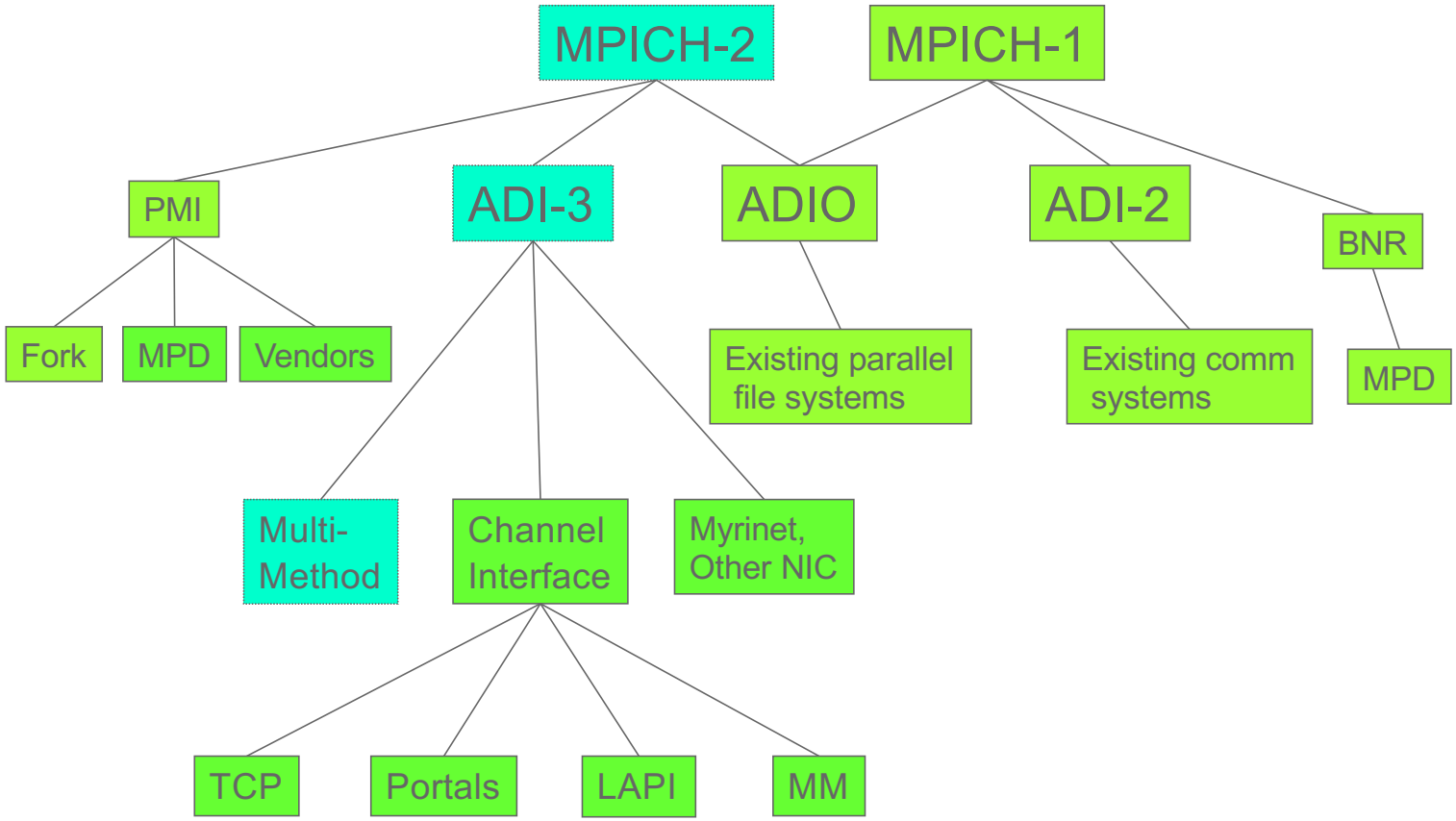
Multilevel Abstract Device Interface

- From the beginning, MPICH used a layered architecture
  - Abstract Device Interface
    - Defined a set of layers
      - Implementing the lowest layer (mostly basic, point-to-point communication and minimal process management/information) sufficient to implement all of MPI
      - Intermediate layers can be replaced, providing additional opportunities for vendor optimization
- Greatest Common Denominator approach
  - Basic level uses a greatest common denominator – capabilities that are universal in distributed memory parallel systems
- Abstraction hides network and implementation specifics

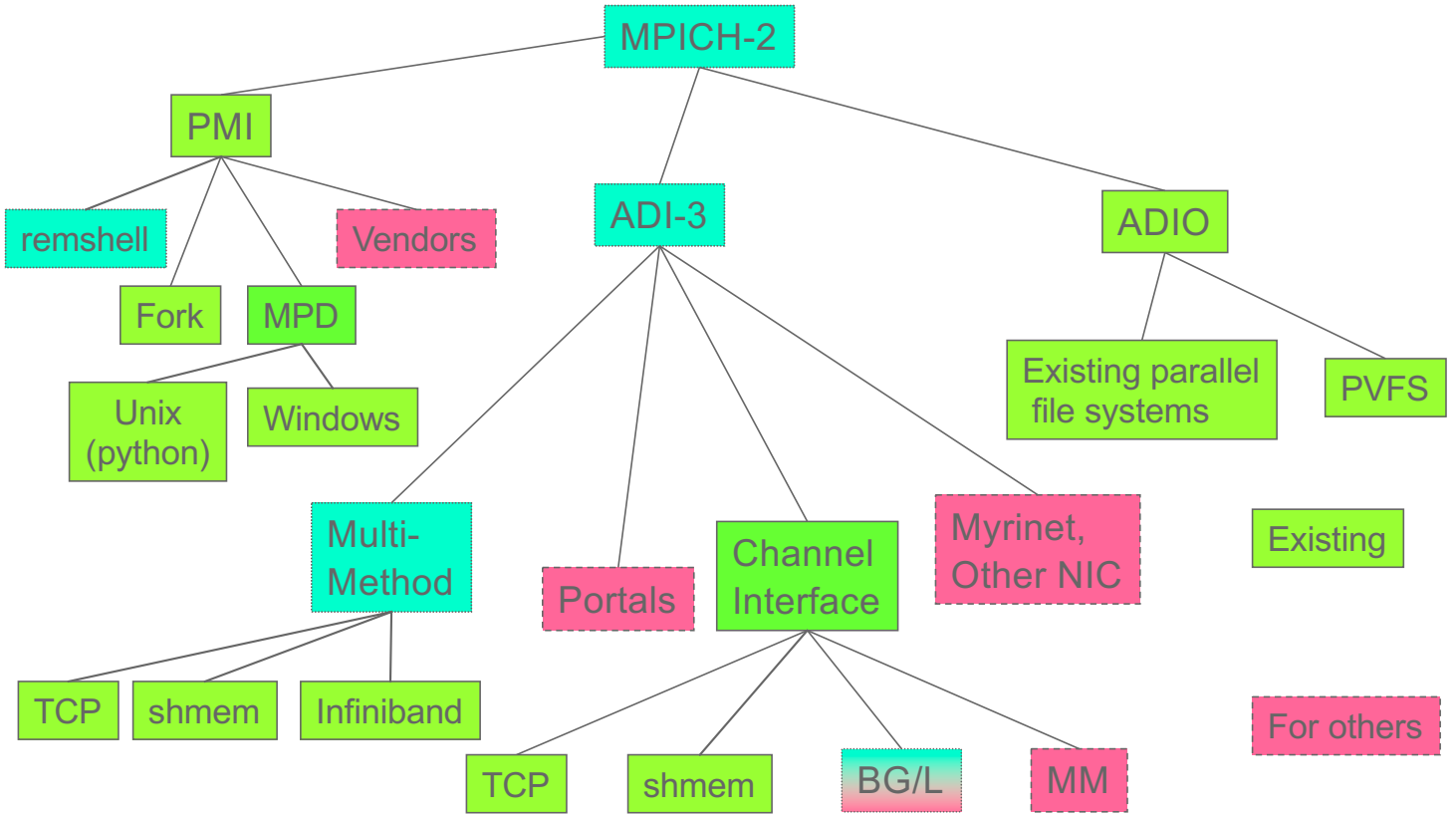
Structure of MPICH(-1)



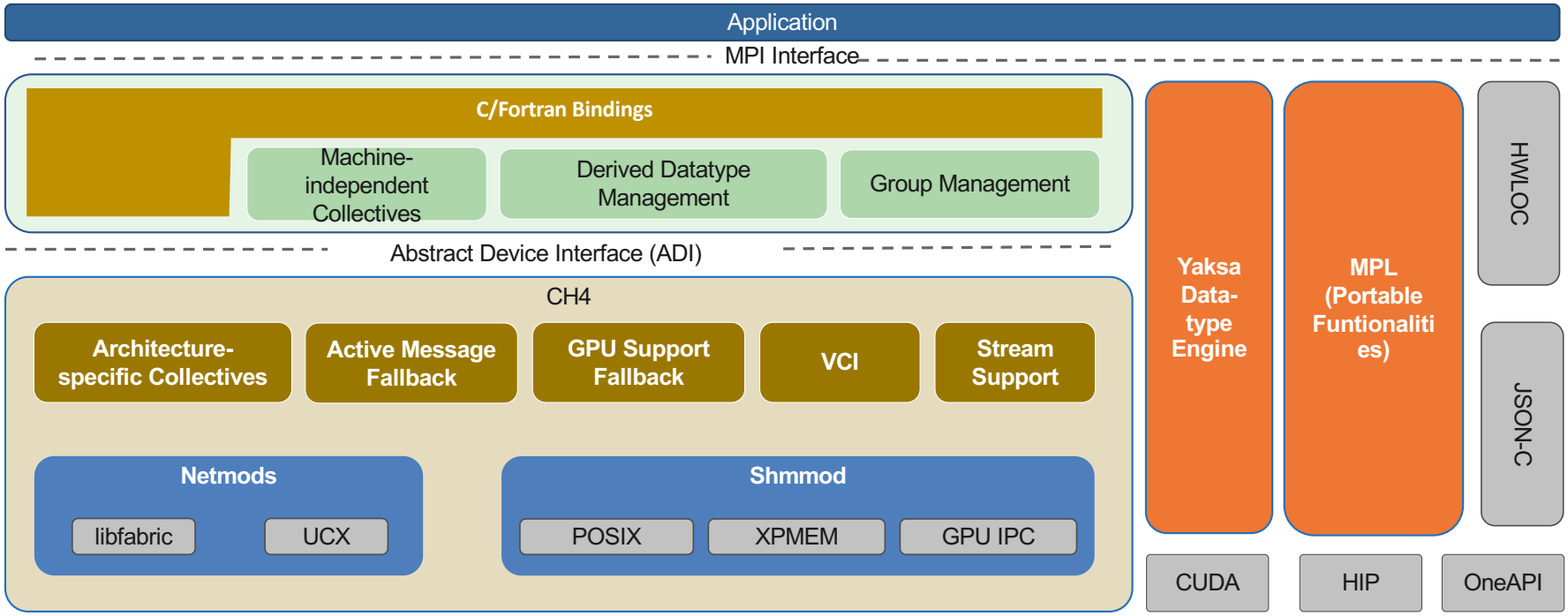
Structure of MPICH-2



Structure of MPICH-2



MPICH Layered Structure



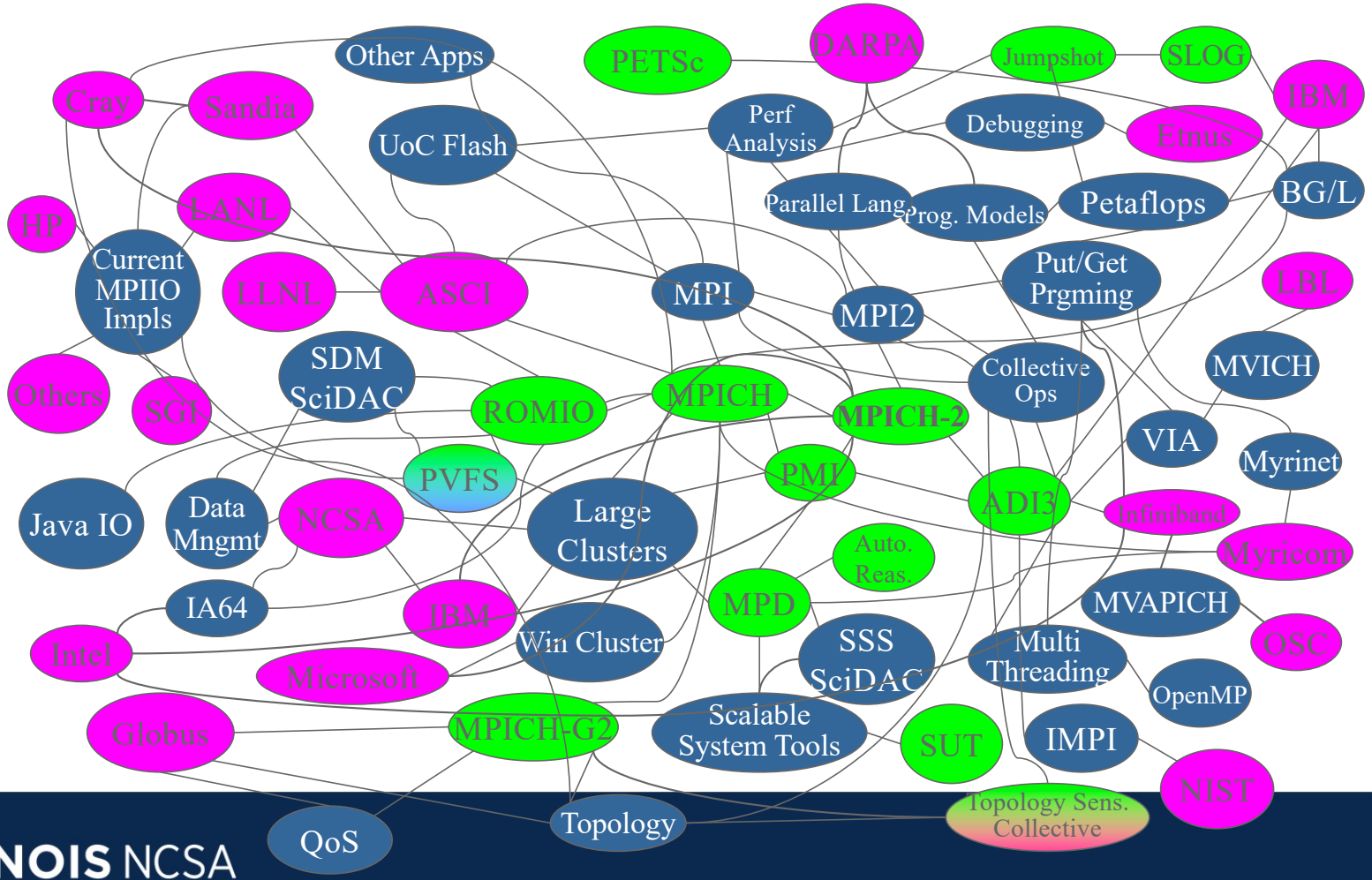
Software Engineering in MPICH

- MPICH began more than 30 years ago
  - Software engineering has evolved substantially
  - At the time MPICH started, “make” was considered exotic (remember Imake?), there was no dominant OS or processor architecture, memory was precious, networks were relatively slow, and source code control was RCS (or email, as in “I’m editing the file now”)
- MPICH also started with a small “team” (me and Rusty), so we had to make the most of what automation we could find

Software Engineering in MPICH – The Early Years

- Program Build Tools
  - Adopt tools when possible
    - Autoconf/configure
      - Early adoption of “capability” based, rather than “system name” based, feature identification
      - Both made it easier to adapt to new systems and to the evolution of existing systems
    - make (not universal when we started!)
    - several generations of bug tracking (email based, some with tcl/tk code)
  - Build ones we needed (or borrow from other projects like PETSc)
    - Doctext creates man pages in nroff and HTML from structured comment in source file
    - Bfort wrapper generator for Fortran
    - MPI wrapper generator
    - Added an alternative to Automake for creating Makefile.in
      - Addressed limitations in automake as well as a different model of incremental build
- W. Gropp and E. Lusk. Sowing MPICH: A case study in the dissemination of a portable environment for parallel scientific computing. The International Journal of Supercomputer Applications and High Performance Computing, 11(2):103–114, Summer 1997.

MPICH2 and the World (circa 2000)



Software Engineering in MPICH – Later Years

- Several generations of automated testing
  - Homegrown systems to start, complete with web interface to results
  - Integrated testing of random combination of features
  - Replaced with community efforts
    - Lost some options (random feature testing) and gained problematic options such as "expected fail"
      - Expected fail needs to be used in a very disciplined manner and reviewed frequently; unfortunately, testing tools don't enforce this. We did suffer some release failures as a result
- Built automated coverage analysis, exploiting gcov and our extensive test suite
  - Identified untested (and incorrect) code
- Built automated coding standards check, including option for signed exceptions

Software Engineering in MPICH

- Performance focus in MPICH and HPC codes a challenge compared with mainstream programming
  - Low latency in particular is a challenge
  - Typical tradeoffs for clarity and maintainability may have performance impacts
    - Which makes them appropriate in some parts of MPICH but not in others
  - Thread safety another point of contention
    - Overly conservative thread safety approaches common because of their safety and simplicity
    - Can have significant performance impact
      - Correctness requires careful attention to memory model, but can be accomplished

MPICH Derived Implementations

- MPICH achieved its goal of being the base for many implementations, including many vendor-specific versions
- Many MPI Implementations started from MPICH, including:
- Vendors
  - Intel MPI
  - Cray MPI
  - NEC MPI (SX4)
  - Microsoft MPI
  - MPI for IBM BlueGene
- MVAPICH

- Many research implementation also used MPICH:
- MPICH-V
  - MPICH-V: toward a scalable fault tolerant MPI for volatile nodes, SC02
- MPICH-MP (Multi-Platform MPICH)
- MPICH-OFI
  - Efficient implementation of MPI-3 RMA over openFabrics interfaces
- MPICH-Phoenix
  - Phoenix: A Runtime Environment for High Performance Computing on Chip Multiprocessors
- MPICH-G2
  - MPICH-G2: A Grid-enabled implementation of the Message Passing Interface
- …

MPICH as a Research Vehicle

- MPICH has been used by the community as a vehicle for research into both MPI implementations and parallel programming systems
- Some examples include
  - Scalable job startup and process management, as well as well-defined interface to process managers
    - PMI, MPIR
  - Efficient implementation of MPI datatypes
  - Efficient implementation of collective communication
  - Efficient implementation of one-sided communication
  - Thread safety
  - Parallel I/O
  - Process topologies
  - Extreme scale (a million or more processes)
  - Fault Tolerance
- MPICH as a research project has helped provide financial support for improving and extending MPICH

MPICH-Related Research over the years

1985-1994

- **Argonne began research efforts to develop portable and high-performance message-passing libraries (p4, Chameleon)**
- 1992: MPI Forum was established to define a standard API for message passing, with Argonne playing a leading role
- **1994: MPI-1 Standard released, along with the first complete implementation of MPI (MPICH)**
- MPICH ran efficiently on all platforms, which led to the widespread adoption of MPI by applications

1995-2007

- Argonne convened and led the MPI-2 Forum
- **MPI-2 Standard released with many important new features**
- Next major version of MPICH (MPICH2) released with support for all of MPI-2
- **Continued research into all aspects of implementing MPI efficiently (datatypes, collectives, RMA, I/O, fault tolerance, etc.) resulting in numerous publications**
- **2005 R&D 100 Award**

2008-2019

- **MPI-3 Standard released in 2012**
- MPICH 3.0 released, which supported all of MPI-3
- **Continued research in efficient RMA, scalable process management, efficient multithreaded MPI communication**
- MPICH supported by ECP to develop scalable MPI for exascale
- Continued codesign with vendor partners (Intel, Cray, IBM, etc.)

2020 -

- All exascale systems use MPICH
- **MPI-4.1 standard released in 2023**
- MPICH 4 supports all of MPI-4
- Used by many vendors (Intel, HPE/Cray, etc.)
- **Enabled scientific discovery through many large-scale parallel applications**
- **MPI-5.0 standard released in 2025**

Some Key MPICH Papers

- W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing, 22(6):789–828, September 1996.
  - The original MPICH paper (3573 citation)
- W. Gropp and E. Lusk. Sowing MPICH: A case study in the dissemination of a portable environment for parallel scientific computing. The International Journal of Supercomputer Applications and High Performance Computing, 11(2):103–114, Summer 1997.
  - Talks about how we built and supported MPICH (61 citations)
- W. Gropp and E. Lusk, "Reproducible measurements of MPI performance characteristics," in Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer-Verlag Lecture Notes in Computer Science #1897, edited by Jack Dongarra, Emelio Luque, and Tomas Margalef, pp. 11-18.
  - High-quality, reproducible point-to-point and collective benchmarks (298 citations)
- Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in MPICH. International Journal of High Performance Computer Applications,19(1):49–66, 2005.
  - Fast, scalable collectives (1297 citations)
- Rajeev Thakur, William Gropp, and Ewing Lusk. Data sieving and collective I/O in ROMIO. In Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation, pages 182–189. IEEE Computer Society Press, February 1999.
  - Key optimization for MPI I/O (747 citations)
- Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing MPI-IO portably and with high performance. In Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems, pages 23–32. ACM Press, May 1999.
  - ROMIO implementation (526 citations)
- Darius Buntinas, Guillaume Mercier, and William Gropp. Design and evaluation of Nemesis, a scalable, low-latency, message-passing communication subsystem. In Stephen John Turner, Bu Sung Lee, and Wentong Cai, editors, Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid2006), pages 521–530, May 2006.
  - The key communication layer in MPICH2/ADI3 (158 citations)
- Translational Research in the MPICH Project
  - https://www.sciencedirect.com/science/article/pii/S1877750320305044 or https://doi.org/10.1016/j.jocs.2020.101203
  - William Gropp, Rajeev Thakur, and Pavan Balaji. Translational research in the MPICH project. Journal of Computational Science, page 101203, 2020.
- More papers at https://www.mpich.org/publications/

Sampling of Best Papers from MPICH Group

- William D. Gropp and Rajeev Thakur. Issues in developing a thread-safe MPI implementation. In Bernd Mohr, Jesper Larsson Träff, Joachim Worringen, and Jack Dongarra, editors, Recent Advances in Parallel Virtual Machine and Message Passing Interface, number LNCS 4192 in Springer Lecture Notes in Computer Science, pages 12–21. Springer, September 2006. **Outstanding Paper Award** (1 of 3).
- Salman Pervez, Ganesh Gopalakrishnan, Robert M. Kirby, Rajeev Thakur, and William D. Gropp. Formal verification of programs that use MPI one-sided communication. In Bernd Mohr, Jesper Larsson Träff , Joachim Worringen, and Jack Dongarra, editors, Recent Advances in Parallel Virtual Machine and Message Passing Interface, number LNCS 4192 in Springer Lecture Notes in Computer Science, pages 30–39. Springer, September 2006. **Outstanding Paper Award** (1 of 3).
- Rajeev Thakur and William Gropp. Test suite for evaluating performance of MPI implementations that support MPI THREAD MULTIPLE. In Cappello et al. 14th European PVM/MPI User's Group Meeting 2008, pages 46–55. **Outstanding paper** (1 of 4).
- Jesper Larsson Träff , William Gropp, and Rajeev Thakur. Self-consistent MPI performance requirements. In Cappello et al. 14th European PVM/MPI User's Group Meeting 2008, pages 36–45. **Outstanding paper** (1 of 4).
- Suren Byna, Yong Chen, W. D. Gropp, Xian-He Sun, and Rajeev Thakur. Parallel I/O prefetching using MPI file caching and I/O signatures. In Proceedings of SC08. IEEE and ACM, 2008. **Best Poster**.
- Suren Byna, Yong Chen, W. D. Gropp, Xian-He Sun, and Rajeev Thakur. Hiding I/O latency with pre-execution prefetching for parallel applications. In Proceedings of SC08. IEEE and ACM, 2008. **Finalist for Best Paper and Best Student Paper**.
- Pavan Balaji, Anthony Chan, Rajeev Thakur, William Gropp, and Ewing Lusk, "Toward Message Passing for a Million Processes: Characterizing MPI on a Massive Scale Blue Gene/P," *Computer Science -- Research and Development*, 24(1-2):11-19, September 2009. (pdf) (**Best Paper Award** at the Int'l Supercomputing Conference (ISC) 2009)
- Paul Sack and William Gropp. Faster topology-aware collective algorithms through non-minimal communication. In Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, PPoPP '12, pages 45–54, New York, NY, USA, 2012. ACM. **Best Paper**.
- Yin Lu, Yong Chen, Prathamesh Amritkar, Rajeev Thakur, and Yu Zhuang, "A New Data Sieving Approach for High Performance I/O," in *Proc. of the 7th Int'l Conference on Future Information Technology (FutureTech'12)*, June 2012. (pdf) (**Best Paper Award**)
- Xin Zhao, Pavan Balaji, William Gropp, and Rajeev Thakur, "Optimization Strategies for MPI-Interoperable Active Messages," in *Proc. of the 13th IEEE Int'l Conference on Scalable Computing and Communication (ScalCom 2013)*, December 2013. (pdf) (**Best Paper Award**)
- Judicael A. Zounmevo, Xin Zhao, Pavan Balaji, William Gropp, and Ahmad Afsahi. Non- blocking epochs in MPI one-sided communication. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '14, pages 475–486, Piscataway, NJ, USA, 2014. IEEE Press. **Best paper finalist.**
- Tarun Prabhu and William Gropp. DAME: A runtime-compiled engine for derived datatypes. In Jack J. Dongarra, Alexandre Denis, Brice Goglin, Emmanuel Jeannot, and Guillaume Mercier, editors, EuroMPI, pages 4:1–4:10. ACM, 2015. **Best paper.**

Changing Computing and MPICH

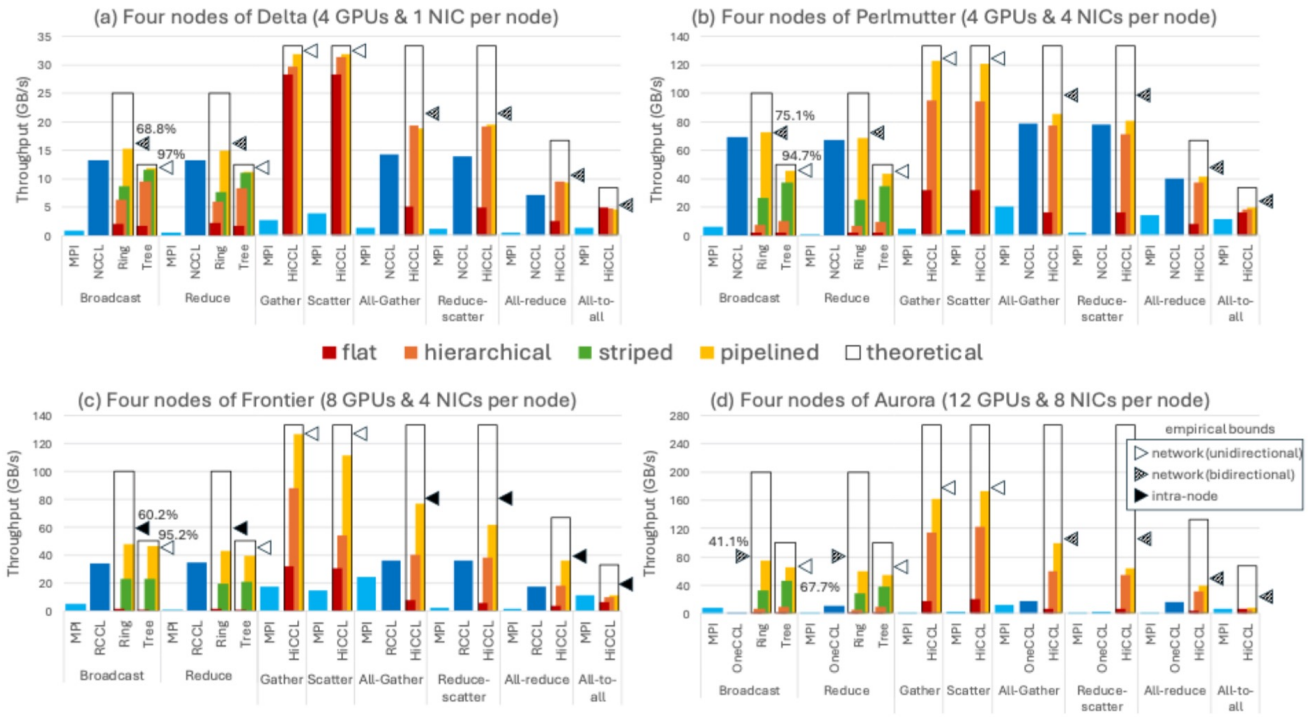
- Over the years, parallel computers have changed dramatically
  - Early systems had multiple chips per node, supporting one thread
  - Nodes (and later chips) added shared memory
  - Networks have gone through many changes, with different emphases on bandwidth, latency, interoperability, …
  - Accelerators have evolved, from ”attached with separate memory” to today’s more integrated memory architecture
  - Number of cores and threads has exploded, making MPI everywhere resource intensive compared to MPI+X or node-aware implementations
  - Process management, scalable startup/rundown, fault tolerance, debugging have all added requirements outside of the MPI standard

Future and Summary

- MPI and MPICH continue to evolve
  - Exploring ideas for expanding MPI and for new MPI features
  - This was MPICH's original purpose – help evaluate possible directions for MPI
- New implementation strategies, algorithms, and environments
  - AI bringing a new community to HPC – with different needs, expectations, and experience
  - An example is GPU aware collectives
    - Must you use NCCL to get good performance
    - Are there MPI features that make it hard to implement fast GPU collectives?
  - Many others, including RMA signaling, streaming, IO, fault tolerance, …

Fast Collectives without NCCL

- Work of Mert Hidayetoglu and colleagues
- GPU aware Collectives, assembled from building blocks
- Hollow rectangles give performance model bounds
- Note allows *portable* construction of GPU aware collectives competitive with NVIDIA NCCL and faster than most MPI implementations
- Boxes show limits on performance



Recent Progress in MPICH

- Harnessing GPUs and Evolving MPICH for AI Applications
- Performance Optimizations for Aurora
  - In close collaboration with ALCF, Intel, and application teams to further optimize GPU communication on Aurora
  - Improving memory scalability w.r.t HBM on Aurora
- Enhancing MPI Collective Communication
  - Automated tuning for selecting the optimal collective algorithm. 1.8x-3x speedup on Aurora
  - Composable collective algorithms to leverage acceleration from GPU collective and external libraries
- Contributed to the ABI (Abstract Binary Interface) in MPI-5 Standard
  - Created the proof-of-concept implementation for MPI-5 ABI that heavily influenced the MPI-5 Standard ABI

Support for Memory Allocation Kinds Side Document

- https://github.com/mpi-forum/mem-alloc
  - Defines CUDA, ROCm, and Level Zero memory kinds
  - Extends mpi and system kinds defined in MPI-4.1

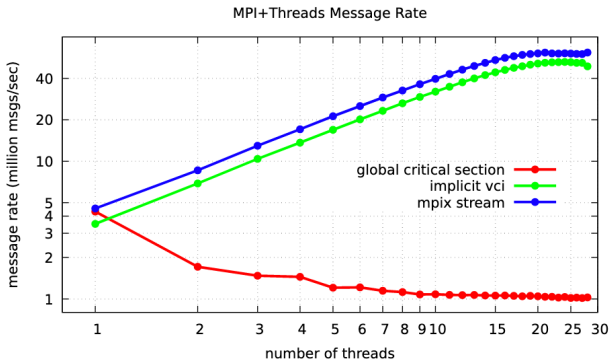
- MPICH supports all three kinds (with or without restrictors) defined in the side document.
  - GPU support remains controlled by the MPIR_CVAR_ENABLE_GPU environment variable. Requesting GPU support is not required (for now).
  - Queries return the kinds supported, e.g. `mpi,system,cuda`. MPICH does not differentiate any restrictors at this time but will return them if requested.

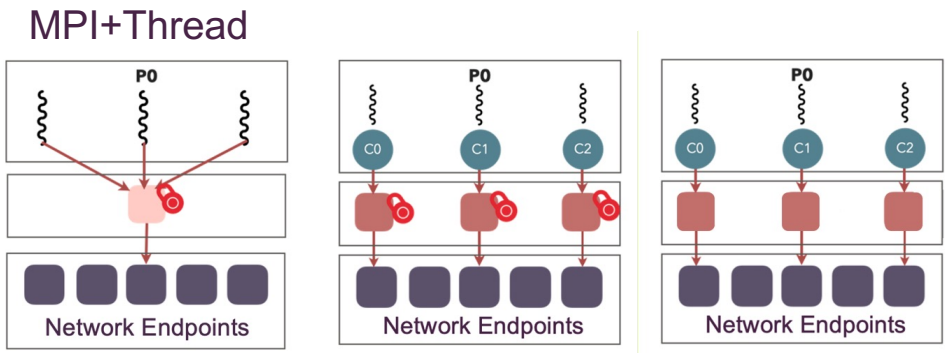
```
/* test if MPI_COMM_WORLD gets the right value */
MPI_Info info;
MPI_Comm_get_info(MPI_COMM_WORLD, &info);
MPI_Info_get(info, "mpi_memory_alloc_kinds", MPI_MAX_INFO_VAL,
             value, &flag);
MPI_Info_free(&cinfo);
if (flag) {
    printf("mpi_memory_alloc_kinds = \"%s\"\n", value);
}
```

```
pmrs-gpu-240-01% mpiexec -n 1 ./memory_alloc_kinds
mpi_memory_alloc_kinds value is "mpi,system,cuda"
pmrs-gpu-240-01% mpiexec -n 1 -memory-alloc-kinds cuda:device ./memory_alloc_kinds
mpi_memory_alloc_kinds value is "mpi,system,cuda,cuda:device"
pmrs-gpu-240-01%
```

MPIX Stream for Hybrid Programming

Problem: MPI does not see execution contexts, e.g. threads





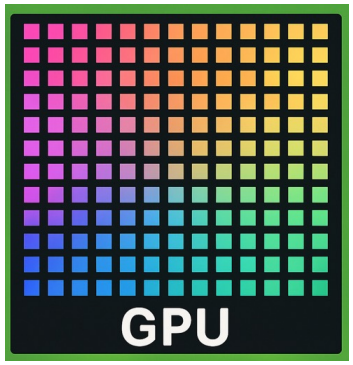

```
int MPIX_Stream_create(MPI_Info info, MPIX_Stream *stream)
```

```
int MPIX_Stream_comm_create(MPI_Comm parent_comm, MPIX_Stream stream,
    MPI_Comm *stream_comm)
```

Zhou, H., Raffenetti, K., Guo, Y., & Thakur, R. (2022, September). MPIX Stream: An explicit solution to hybrid MPI+ X programming. In *Proceedings of the 29th European MPI Users' Group Meeting*.

Enqueue communications

Problem: MPI does not see offloading execution context

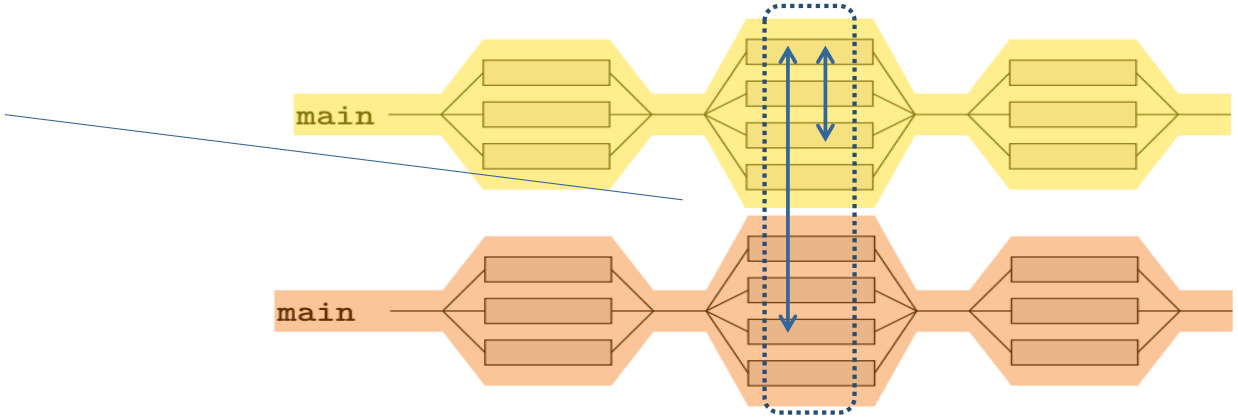



```
int MPIX_Send_enqueue(buf, count, datatype, dest, tag, comm)
int MPIX_Recv_enqueue(buf, count, datatype, source, tag, comm, status)
int MPIX_Isend_enqueue(buf, count, datatype, dest, tag, comm, request)
int MPIX_Irecv_enqueue(buf, count, datatype, source, tag, comm, request)
int MPIX_Wait_enqueue(request, status)
int MPIX_Waitall_enqueue(count, array_of_requests, array_of_statuses)
```

MPI × Threads

Problem: threads cannot use MPI

Unified Parallel Region



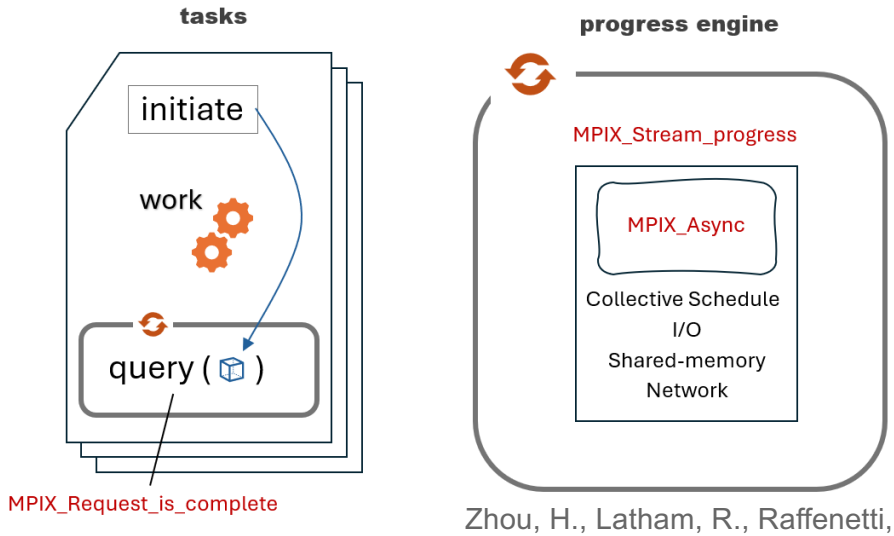

```
#pragma omp parallel {
    MPIX_Threadcomm_start(threadcomm);
    /* use threadcomm within parallel region */
    MPIX_Threadcomm_finish(threadcomm);
}
```

Zhou, H., Raffenetti, K., Zhang, J., Guo, Y., & Thakur, R. (2023, September). Frustrated With MPI+Threads? Try MPIxThreads!. In *Proceedings of the 30th European MPI Users' Group Meeting*

Extensions on MPI Progress

Problem: MPI implicitly makes progress

```
MPI_Isend/Irecv
[computation]
MPI_Waitall
```




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

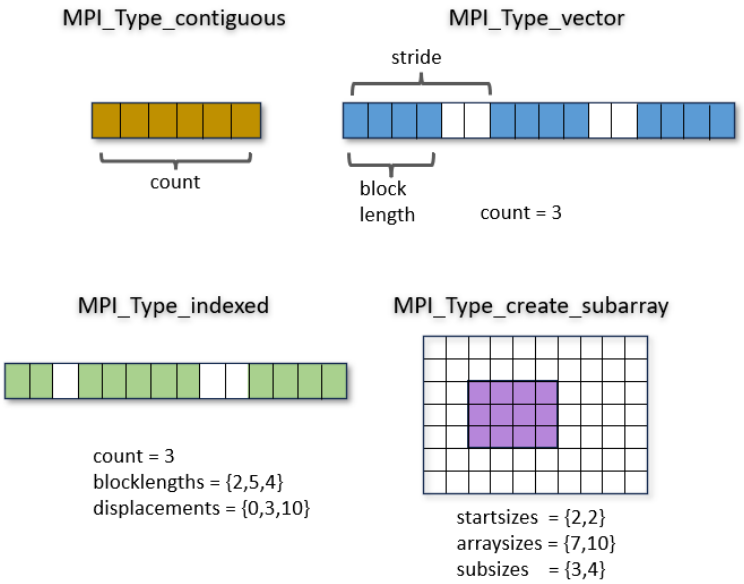
```
int MPIX_Async_start(MPIX_Async_poll_function poll_fn,
                     void *extra_state,
                     MPIX_Stream stream)
```

```
int MPIX_Request_is_complete(MPI_Request request)
```

Zhou, H., Latham, R., Raffenetti, K., Guo, Y., & Thakur, R. (2024, November). MPI Progress For All. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 425-435). IEEE.

Use MPI just for its datatype

Problem: MPI datatypes only can be used by MPI




```
int MPIX_Type_iov_len(MPI_Datatype datatype,
    MPI_Count max_iov_bytes,
    MPI_Count *iov_len,
    MPI_Count *actual_iov_bytes)
```

```
int MPIX_Type_iov(MPI_Datatype datatype,
                  MPI_Count iov_offset,
                  MPIX_Iov *iov,
                  MPI_Count max_iov_len,
                  MPI_Count *actual_iov_len)
```

Zhou, H., Raffenetti, K., Guo, Y., Gillis, T., Latham, R., & Thakur, R. (2024). Designing and prototyping extensions to the Message Passing Interface in MPICH. *The International Journal of High Performance Computing Applications*, *38*(5), 527-545.

Some Lessons from MPICH

- Automate code creation and management as much as possible
  - Note AI tools are revolutionizing software engineering
- Design for performance and portability
  - Parallel computing is about performance
  - Design to allow customization
  - You can build (trans)portable code by identifying a hierarchy of abstractions, isolating non-portable features
- Be willing to rethink your design and make large-scale changes
  - MPICH has had multiple major changes, some in components and some in the overall architecture
- Build for others and *Listen to your users*