

# The Message Passing Interface: Towards MPI 4.0 & Beyond

*Martin Schulz*  
*TU Munich*  
*Chair of the MPI Forum*

MPI Forum BOF @  
ISC 2019



<https://www.mpi-forum.org/>

# The MPI Forum Drives MPI



## **Standardization body for MPI**

- Discusses additions and new directions
- Oversees the correctness and quality of the standard
- Represents MPI to the community

## **Organization consists of chair, secretary, convener, steering committee, and member organizations**

## **Open membership**

- Any organization is welcome to participate
- Consists of working groups and the actual MPI forum (plenary)
- Physical meetings 4 times each year (3 in the US, one with EuroMPI/Asia)
  - Working groups meet between forum meetings (via phone)
  - Plenary/full forum work is done mostly at the physical meetings
- Voting rights depend on attendance
  - An organization has to be present two out of the last three meetings (incl. the current one) to be eligible to vote

# The Bulk of Work is in the Working Groups



## **Collective Communication, Topology, Communicators, Groups**

- Torsten Hoefler, Andrew Lumsdaine and Anthony Skjellum

## **Fault Tolerance**

- Wesley Bland, Aurélien Bouteiller and Rich Graham

## **HW Topologies**

- Guillaume Mercier

## **Hybrid Programming**

- Pavan Balaji and Jim Dinan

## **Big Count**

- Jeff Hammond and Anthony Skjellum

## **Persistence**

- Anthony Skjellum

## **Point to Point Communication**

- Rich Graham and Dan Holmes

## **Remote Memory Access**

- Bill Gropp and Rajeev Thakur

## **Semantic Terms**

- Rolf Rabenseifner and Purushotham Bangalore

## **Sessions**

- Dan Holmes

## **Tools**

- Kathryn Mohror and Marc-Andre Hermanns

# The Bulk of Work is in the Working Groups



## **Collective Communication, Topology, Communicators, Groups**

- Torsten Hoefler, Andrew Lumsdaine and Anthony Skjellum

## **Fault Tolerance**

- Wesley Bland, Aurélien Bouteiller and Rich Graham

## **HW Topologies**

- Guillaume Mercier

## **Hybrid Programming**

- Pavan Balaji and Jim Dinan

## **Big Count**

- Jeff Hammond and Anthony Skjellum

## **Persistence**

- Anthony Skjellum

## **Point to Point Communication**

- Rich Graham and Dan Holmes

## **Remote Memory Access**

- Bill Gropp and Rajeev Thakur

## **Semantic Terms**

- Rolf Rabenseifner and Purushotham Bangalore

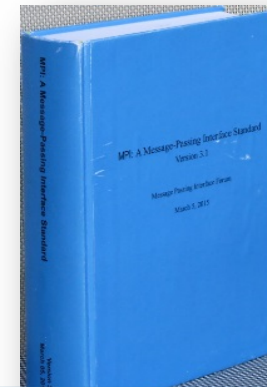
## **Sessions**

- Dan Holmes

## **Tools**

- Kathryn Mohror and Marc-Andre Hermanns

# The Status of MPI



## **MPI 3.0 ratified in September 2012**

- Major new functions

## **MPI 3.1 ratified in June 2015**

- Minor updates and additions

## **Fully adopted in all major MPIs**

## **MPI 4.0 work coming to an end**

- Target end of 2020 or early 2021

## **Likely new features**

- New init options via MPI Sessions
- New tool interface for events
- More optimization potential via persistence and non-blocking operations
- Solution for “Big Count” via some form of overloading
- Simple fault handling for P2P operations
- Introduction of assertions
- Topology optimizations

## **Draft standard available as of 11/18, second one planned for 11/19**

- Standards and drafts available at <http://www.mpi-forum.org/>

Available through HLRS  
-> MPI Forum Website



# How New Features Get Added to MPI

1. New items brought to a matching working group for discussion
2. Creation of preliminary proposal
3. Socializing of idea driven by the WG  
Through community discussions, user feedback, publications, ...  
  
Development of full proposal  
In many cases accompanied with prototype development work
4. MPI forum reading/voting process  
One reading  
Two votes  
Slow and consensus driven process
5. Once enough topics are completed:  
Publication of a new standard



# Participate!!!



## **MPI Forum is an open forum**

- Everyone / every organization can join
- Voting rights depends on attendance of physical meetings

## **Major initiatives towards MPI 4.0**

- Active discussion in the respective WGs
- Need/want community feedback
- Feature freeze this year meetings
- Target date: late 2020, early 2021

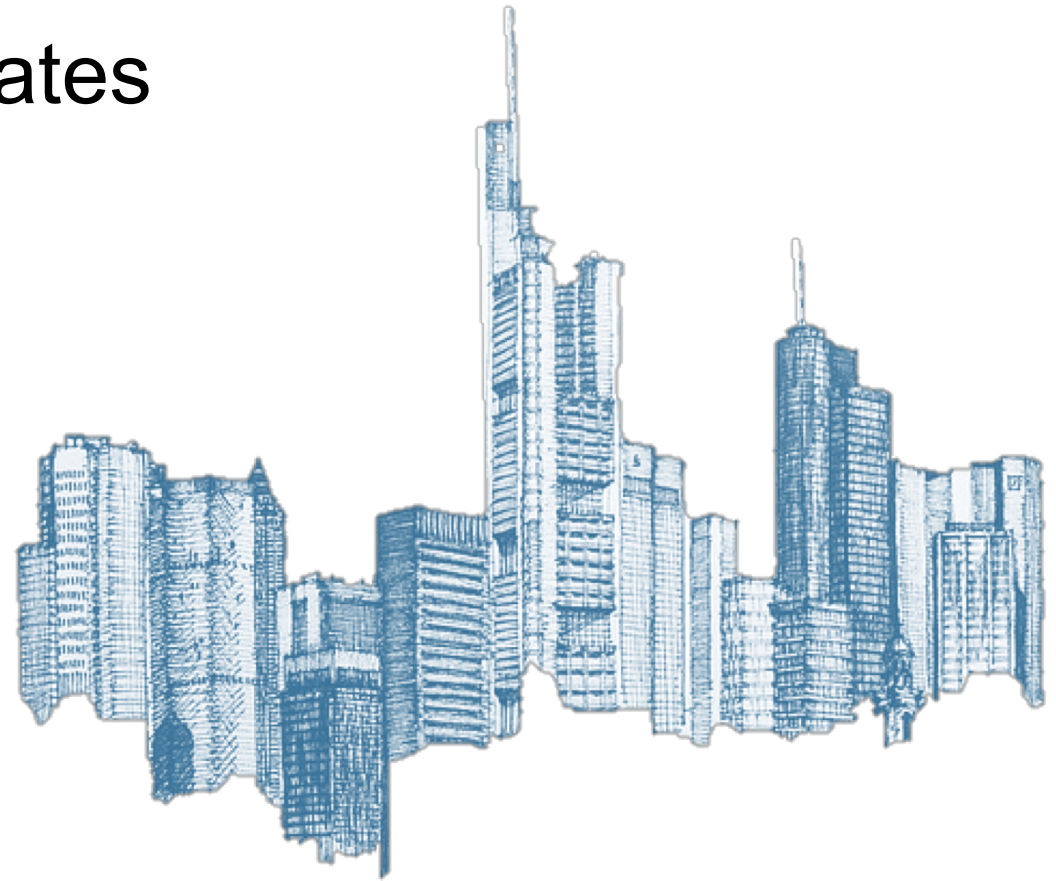
## **Get involved**

- Let us know what you or your applications need
  - [mpi-comments@mpi-forum.org](mailto:mpi-comments@mpi-forum.org)
- Participate in WGs
  - Email list and Phone meetings
  - Each WG has its own Wiki
- Join us at a MPI Forum F2F meeting
  - Next meetings: Zürich, CH (Sep.), Albuquerque, NM/USA (Dec.)
- EuroMPI will also be in Zürich: Sep. 10-13, 2019

<http://www.mpi-forum.org/>

# The Message Passing Interface: Towards MPI 4.0 & Beyond

## Technical MPI 4.0 Updates



<https://www.mpi-forum.org/>



# Technical Updates



## **Dan Holmes, EPCC**

- MPI Sessions

## **Marc-Andre Hermanns, RWTH Aachen**

- New Interfaces for MPI Tools

## **Anthony Skjellum, U. of Tennessee at Chattanooga**

- Persistent Collectives
- Big Count
- Towards new language interfaces

# SESSIONS WG

---

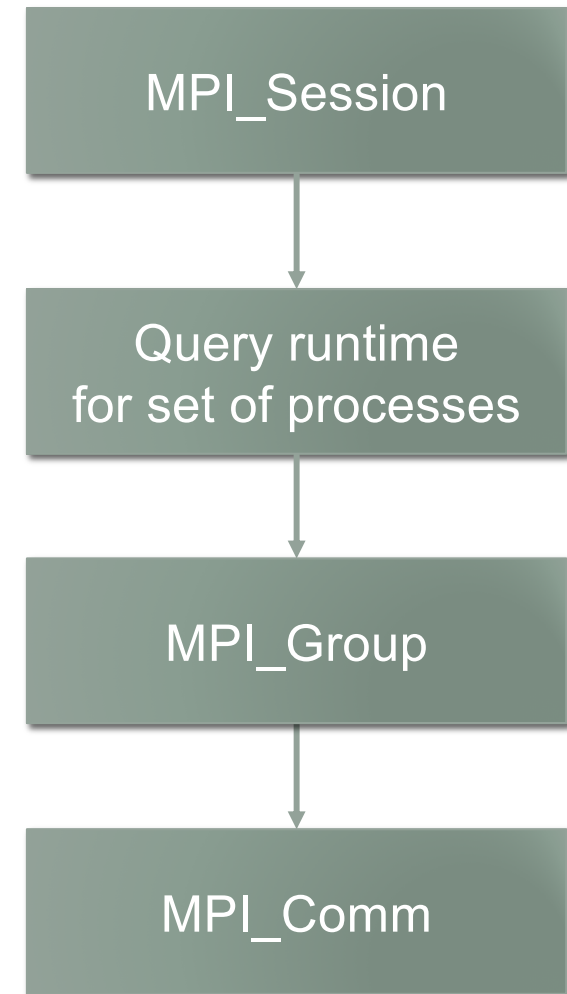
Dan Holmes

# What are sessions?

- A simple local handle to the MPI library
- An isolation mechanism for interactions with MPI
- An extra layer of abstraction/indirection
- A way for MPI/users to interact with underlying runtimes
  - Schedulers, resource managers, others
- A solution for some threading problems in MPI
  - Thread-safe initialisation by multiple entities (e.g. libraries)
  - Re-initialisation after finalisation
- A way to avoid some implementation headaches in MPI
  - Implementing `MPI_COMM_WORLD` efficiently is hard
- An attempt to control the error behaviour of initialisation

# How can sessions be used?

- Initialise a session
- Query available process “sets”
- Obtain info about “sets” (optional)
- Create an MPI\_Group directly from a “set”
- Modify the MPI\_Group (optional)
- Create an MPI\_Communicator directly from the MPI\_Group (without a parent communicator)
  - ~~Any type, e.g. cartesian or dist\_graph~~



# Why are sessions a good idea?

- Any thread/library/entity can use MPI whenever it wants
- Error handling for sessions is defined and controllable
- Initialisation and finalisation become implementation detail
- Scalability (inside MPI) should be easier to achieve
- Should complement & assist endpoints and fault tolerance

# Who are sessions aimed at?

- Everyone!
- Library writers: no more reliance on main app for correct initialisation and provision of an input MPI\_Communicator
- MPI developers: should be easier to implement scalability, resource management, fault tolerance, endpoints, ...
- Application writers: MPI becomes 'just like other libraries'

# Sessions WG

- Fortnightly meetings, Monday 1pm Eastern US webex
  - All welcome!
- <https://github.com/mpiwg-sessions/sessions-issues/wiki>
- Future business:
- Dynamic “sets”? Shrink/grow – user-controlled/faults?
- Interaction with tools? Issues caused by isolation?
- Different thread support levels on different sessions?

# Things that got removed

- No more attributes on sessions
  - The special attributes have been moved to communicators
  - Get `MPI_TAG_UB` from the first communicator for the session
- No more “flags” for `MPI_Session_init`
  - Thread support level can be specified using `MPI_Info` parameter
  - It can be queried using the `MPI_Info` object for the session



# Things that got added

- New initial error handler
  - Added by the error handling and fault tolerance WG
  - Currently initial and default error handler is `ERRORS_ARE_FATAL`
  - From MPI-4.0, the user will be able to change this before `MPI_INIT`
  - The Sessions Model uses those error handlers as well
  - Errors before initialisation of MPI can be caught and handled
- Additional always thread-safe functions
  - From MPI-4.0, `MPI_Info` is safe to use before MPI initialisation
  - It is needed to call `MPI_Session_init`
  - It is also needed for the new `MPI_T` events (see next presentation)
- There is now a prototype/reference implementation
  - Written by Nathan Hjelm, targeting Open MPI
  - [https://github.com/hpc/ompi/tree/sessions\\_new](https://github.com/hpc/ompi/tree/sessions_new)



# Current Topics of the MPI Forum Tools WG

Marc-André Hermanns, Kathryn Mohror, Martin Schulz

# Overview of Topics

---

- MPI Info everywhere
- MPI\_T Events
- MPI\_T Universal IDs
- Re-vamping PMPI

# MPI Info Everywhere

---

- MPI Info objects
  - Key/value store
  - Communicate information between user and implementation
- Some info needs to be communicated prior to (MPI) initialization
  - MPI\_T events: event type query, handle allocation, callback registration
- Also needed in other concepts (sessions, resiliency)
- Goal: Inclusion in MPI 4.0 (API mostly stable now)

# MPI\_T Events: Motivation

---

- MPI provides several tool interfaces
- PMPI interface
  - Intercept calls into the MPI library
  - MPI implementation is blackbox
- MPI\_T interface
  - Library implementations expose internal information
  - Library-specific software counters
  - Tools poll for information
    - Information is aggregated
- Access to per-instance information of the MPI library is still missing

# MPI\_T Events: Overview

---

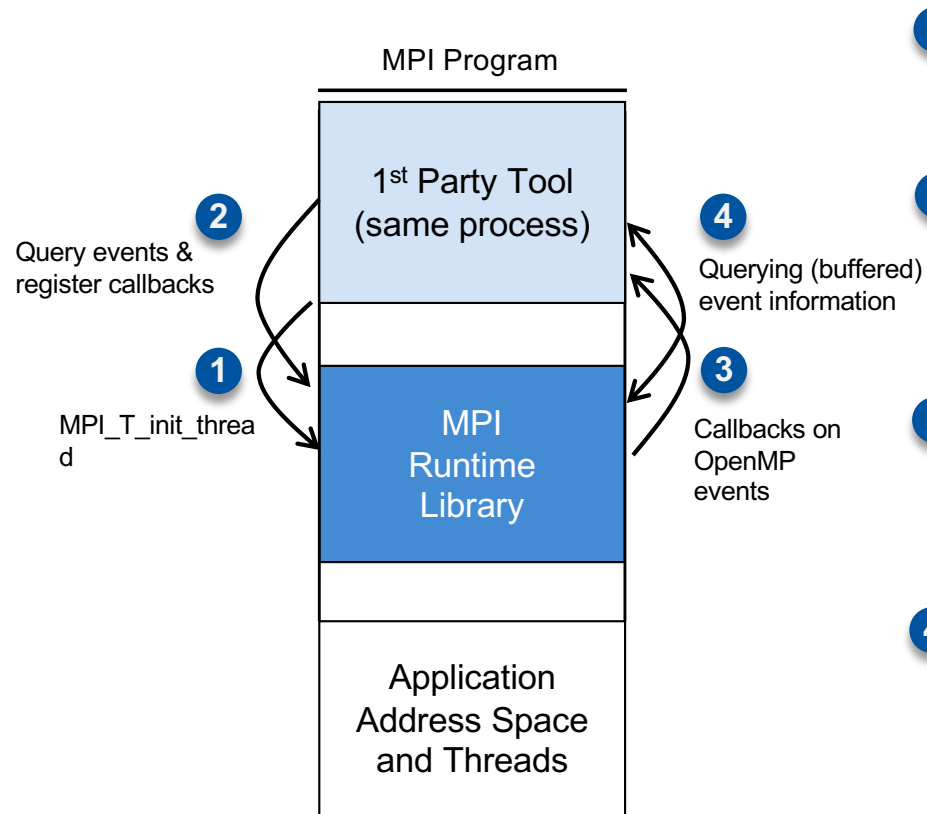
- Blend into existing MPI\_T interface
  - Standardized query interface
  - Implementation choose freely what information to expose
- Access to per-instance information
  - Events represent state changes within the MPI implementation
  - Tools can register callback functions
    - Implementation will invoke callback functions to convey information
- Goal: Inclusion in MPI 4.0 (API mostly stable now)

# MPI\_T Events: Key Features

---

- Transparent buffering
  - Timestamps as part of event meta data
  - Separate event observation from reporting
- Dynamic Safety Requirements
  - Tool can register callbacks with different safety guarantees
  - Runtime most permissive, safe callback
    - Runtime provides specific safety requirement to callback
- Multiple Event Sources
  - Virtual entity as event data provider
  - Enables support for partial ordering requirements

# MPI\_T Events: Workflow



## 1 Runtime initialization

- Tool initializes interface

## 2 Tool initialization

- Tool queries available events
- Tool creates registration handle
- Tool registers callback functions

## 3 Callback functions

- Runtime invokes callback functions
  - Observation time  $\neq$  Invocation time
- Runtime provides event instance handle

## 4 Query event information

- Tool queries event data within callback
- Runtime provides potentially buffered event information



## MPI\_T Universal IDs

---


- MPI\_T does not mandate specific variables to be present for an implementation
- Name/Semantic matching is allowed to change across implementations
- Universal IDs (UIDs) may mitigate problems for portable tools
  - Fixed for specific semantics
    - Changes in semantics imply a new UID
    - UIDs may be shared across implementations
  - Queried separate from name
    - Name/Semantic match remains flexible
  - Namespaced (Separate Vendor Prefixes to UID)
- Goal: Inclusion into one of the next MPI versions (API still in flux)

## Re-vamping PMPI (Codename: QMPI)

---

- Replace existing PMPI interface
  - Support hierarchy of multiple intercepting tools
  - Transparent handle conversion (Tools only use C interface/handles)
- Callback-driven
  - Query function pointer for 'next level' at runtime/initialization time
  - Dynamic registration/deregistration of tools
- Goal: Inclusion in a future MPI version (API still in flux)

**Thank you for your attention**



# MPI-4 Topics

- 1) Persistent Collectives
- 2) Partitioned Point-to-point
- 3) "Big MPI"

Anthony Skjellum, PhD  
University of Tennessee at Chattanooga  
tony-skjellum@utc.edu

June 19, 2019  
ISC 2019

# Outline

- Persistent Collectives
- Partition Communication – Point-to-point
- Big MPI

# Collaborators

- Persistent WG

  - Collective WG

    - Tony Skjellum, Dan Holmes, Ryan Grant, Puri Bangalore, ...

- Point-to-point WG

  - Dan Holmes, Ryan Grant, et al

- Large Count WG

  - Jeff Squyres, Dan Holmes, Puri Bangalore,  
Martin Rüfenacht, Tony Skjellum

- Language Binding Chapter – Puri et al

- Cross-cutting discussions on-going currently

# Persistent Collective Operations

- Use-case: a collective operation is done many times in an application
- The specific sends and receives represented never change (size, type, lengths, transfers)
- A persistent collective operation can take the time to apply a heuristic and choose a faster way to move that data
- Fixed cost of making those decisions could be high but can be amortized over all the times the operation is used
- Static resource allocation can be done
- Choose fast(er) algorithm, take advantage of special cases
- Reduce queueing costs
- Special limited hardware can be allocated if available
- Choice of multiple transfer paths could also be performed

# Basics

- Mirror regular nonblocking collective operations
- For each nonblocking MPI collective, add a persistent variant
- For every MPI\_I<coll>, add MPI\_<coll>\_init
- Parameters are identical to the corresponding nonblocking variant – plus additional MPI\_INFO parameter
- All arguments “fixed” for subsequent uses
- Persistent collective operations cannot be matched with blocking or nonblocking collective calls



# Example

## *Nonblocking collectives API*

```
for (i = 0; i < MAXITER; i++) {  
    compute(bufA);  
    MPI_Ibcast(bufA, ..., rowcomm, &req[0]);  
    compute(bufB);  
    MPI_Ireduce(bufB, ..., colcomm, &req[1]);  
    MPI_Waitall(2, req, ...);  
}
```

## *Persistent collectives API*

```
MPI_Bcast_init(bufA, ..., rowcomm, &req[0]);  
MPI_Reduce_init(bufB, ..., colcomm, &req[1]);  
for (i = 0; i < MAXITER; i++) {  
    compute(bufA);  
    MPI_Start(req[0]);  
    compute(bufB);  
    MPI_Start(req[1]);  
    MPI_Waitall(2, req, ...);  
}
```

# Init/Start

- The init function calls only perform initialization; do not start the operation
- Ex: MPI\_Allreduce\_init
  - Produces a persistent request (not destroyed by completion)
- Requests work with MPI\_Start/MPI\_Startall
- Only inactive requests can be started
- MPI\_Request\_free can free inactive requests

# Ordering of Inits and Starts

- Inits must be ordered like all other collective operations
- Persistent collective operations can be started in the same order, or different orders, at all processes
- MPI\_Startall can contain multiple operations on the same communicator due to ordering freedom
- A new communicator INFO key will be added that asserts persistent collectives starts will be strictly ordered
- In some cases, this may improve performance
- NB: INFO key incompatible with starting multiple persistent collective operations using MPI\_Startall

# Standardization of Persistence - Approved for MPI-4

- <https://github.com/mpi-forum/mpi-issues/issues/25>
- Ticket #25 approved for MPI-4 in September 2018 (Barcelona)
- Ancillary matters to be studied
- <https://github.com/mpi-forum/mpi-issues/issues/83>
- Ticket #83 – to be re-read in September (Zurich)
- <https://github.com/mpi-forum/mpi-issues/issues/90>
- Ticket #90 clarifies text throughout the standard properly to introduce “persistence” in several places where it is not fully mentioned or documented order – to be read again in September, 2019
- “Big MPI” could impact the total API defined here still.

# Partitioned Communication: Allow for Better Thread Parallelism in MPI



- Concept of many actors (threads) contributing to a larger operation in MPI
  - Same number of messages as today!
  - No new ranks/naming of threads---threads can remain remotely anonymous
- Example: many threads work together to assemble a message
  - MPI only has to manage knowing when completion happens
  - These are actor/action counts, not thread level collectives, to better enable tasking models
- No heavy MPI thread concurrency handling required
  - Leave the placement/management of the data to the user
  - Knowledge required: number of workers, which is easily available
- Added Benefit: Match well with Offloaded NIC capabilities
  - Use counters for sending/receiving
  - Utilize triggered operations to offload sends to the NIC



- Expose the “ownership” of a buffer as a shared to MPI
- Need to describe the operation to be performed before contributing segments
- MPI implementation doesn't have to care about sharing
  - Only needs to understand how many times it will be called
- Threads are required to manage their own buffer ownership such that the buffer is valid
  - The same as would be done today for code that has many threads working on a dataset (that's not a reduction)
- Result: MPI is thread agnostic with a minimal synchronization overhead (atomic\_inc)
  - Can alternatively use task model instead of threads, IOVEC instead of contiguous buffer
- Can also be used single-threaded with pipelining only with a strong-progress MPI

# Example for Persistence

39



- Like persistent communications, setup the operation

```
int MPIX_Partitioned_send_init(void *buf, int count, MPI_Datatype data_type,  
    int to_rank, int to_tag, int num_partitions, MPI_Info info, MPI_Comm comm,  
    MPI_Request *request);
```

- Start the request

```
MPI_Start(request)
```

- Add items to the buffer

```
#omp parallel for ...
```

```
int MPIX_Pready( void* buf, int count, MPI_Datatype in_datatype,  
    int offset_index, MPI_Request *request);
```

- Wait on completion

```
MPI_Wait(request)
```

- Optional: Use the same partitioned send over again

```
MPI_Start(request)
```

# Works for non-Persistent Comms

40



- We can have similar functionality with a

```
int MPIX_Ipsend( void *buf, int count, MPI_Datatype data_type, int
to_rank, int to_tag, int num_partitions, MPI_Info info, MPI_Comm
comm, MPI_Request *request);
```

- Works just like a regular send with contribution counts
- First thread to reach Psend gets a request handle back that can be shared with other threads – some MPI locking
- Setup happens on first call
- Track by comm and buff addr
- MPI\_Psend (completing): requires multithreaded MPI program to add buffers via Pready...



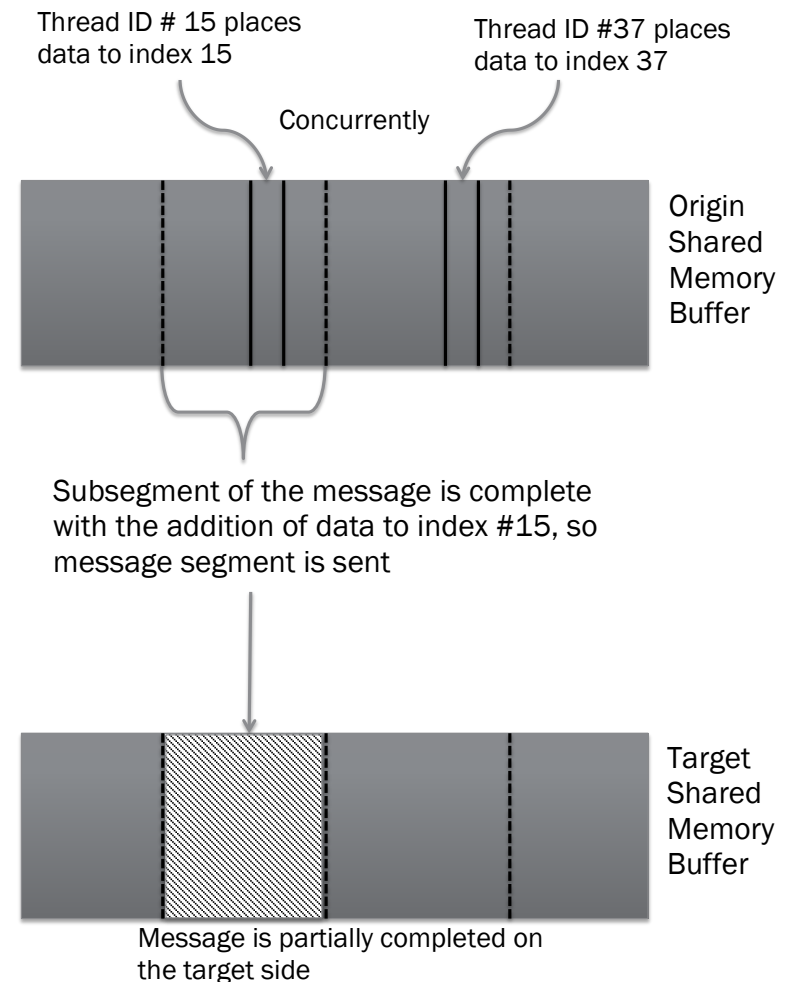
# Opportunities for Optimization

41



MPI implementations can optimize data transfer under the covers:

- Subdivide larger buffers and send data when ready
- Could be optimized to specific networks (MTU size)
- Number of messages will be:  
 $1 < \#messages \leq \#threads/tasks$   
For a partition with 1 part per thread
- Reduces the total number of messages sent, decreasing matching overheads



# Different Approach to Threading

42



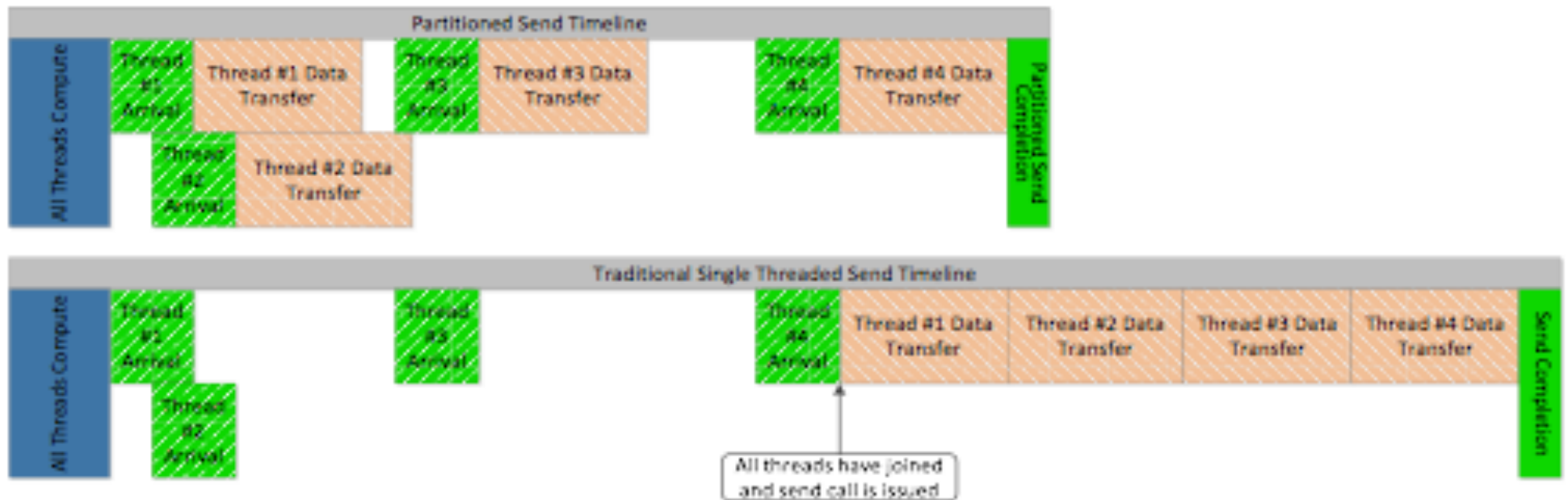
- Partitioned buffer operations can always be considered as multi-threaded
- Using partitioned sends doesn't necessarily require locking in other parts of the MPI library – confine threading in MPI
- Technically, using partitioned buffers would work with `MPI_THREAD_SERIALIZED`
  - If only using partitioned sends, no need for locking in the library
- No more `thread_multiple`?
  - Not quite, but we can have alternative threading modes for MPI, where user management of data is guaranteed and only inherently thread safe operations are called, `MPI_THREAD_PARTITIONED`

# New Type of Overlap

43



- “Early bird communication”
- Early threads can start moving data right away
- Could implement using RDMA to avoid message matching



# Partitioned Communication Benefits

44



- Performance benefit of early-bird overlap
  - Better than current fork-join-communicate methods
- Lightweight thread synchronization
  - Single atomic
- Same message/matching load as today
  - Avoid the coming storm
- A great way to adapt code to use RDMA underneath
  - Keeping existing send/rcv completion semantics
- Easy to translate code
  - May be able to automatically translate send/rcv with simple parallel loops over to psend-type operations
- Read our ISC 2019 paper!

# Standardization of “Partitioned Point-to-point”

- <https://github.com/mpi-forum/mpi-issues/issues/136>
- Vote and/or Reading in September, 2019 (Zurich)

# Big MPI

- Idea: Finally make MPI fully 64-bit clean
- More 2 Gi element transfers “is a pain”
- Solve issue across API: Collectives, Point-to-point, I/O, [and RMA]
- Started with a significant study and prototyping effort by Jeff Hammond that yielded a proposal for Collective communication
  - Workarounds possible for pt2pt do not work well for collectives
  - v/w-collectives and reductions have the most concerns
- Neighborhood collectives fixed the large-displacement problem (for these new ops)
- We have several solution paths...

## Approach 1 - well understood, on-hold

- Add `_Y` to all functions that need extended parameters (formerly `_X`)
- `int` -> `MPI_Count` for the number of datatypes transferred
- `int` for displacements becomes `MPI_Aint` (maybe `MPI_Count`)
- Impact: 133 APIs in MPI-3.1
- Solves: C/C++, FORTRAN
- Works with PMPI fine
- Forum feedback: put on hold, look for other approaches

## Approach 2 - Currently preferred, being worked

- Utilize function polymorphism in C, C++, and FORTRAN
- The 32-bit-clean and 64-bit clean APIs can match across the group, as long as the all the type signatures are compatible with the APIs used
- Adds no apparent new functions to MPI
- Underneath, still have 133+ more functions
- PMPI symbols are still literally defined with `_Y` in the names (or such)
- Controversy on C11 generics still open
- C++ support requires a different header or minimal C++ separate support
- FORTRAN good with modules



## Approach 3 - Currently speculative, being worked

- Complete specification of partitioned communication
- Partitioned APIs are defined --- over time--- for point-to-point, collective, supporting nonblocking, blocking, and persistent, I/O
- Sub-approach #a:
  - Simply change int -> MPI\_Count, and allow MPI\_Aint displacements in all the new APIs
- Sub-approach #b:
  - Simply change int -> MPI\_Count and use partitioned model for large transfers
- MPI-4.0 -> Only point-to-point solved
- MPI-4.x, x>=2 -> Standardize the rest of partitioned comms, and solve rest of Big MPI thusly over time; merges with persistent collective extensions too

# Standardization of “Big MPI”

- New ticket posed in Chicago, May 2019
- Overrides previous (\_X/\_Y) tickets:
  - <https://github.com/mpi-forum/mpi-issues/issues/137>
- Virtual Meeting, July 24
- Reading in September, 2019 (Zurich)

Thank You!  
😊

**Contact:**

[tony-skjellum@utc.edu](mailto:tony-skjellum@utc.edu)

