"MPI + OpenMP" to "MPI in Harmony with OpenMP"

Developments in the MPI Standard and how they could interact with OpenMP

Martin Schulz
Chair for Computer Architecture and Parallel Systems
Technical University of Munich

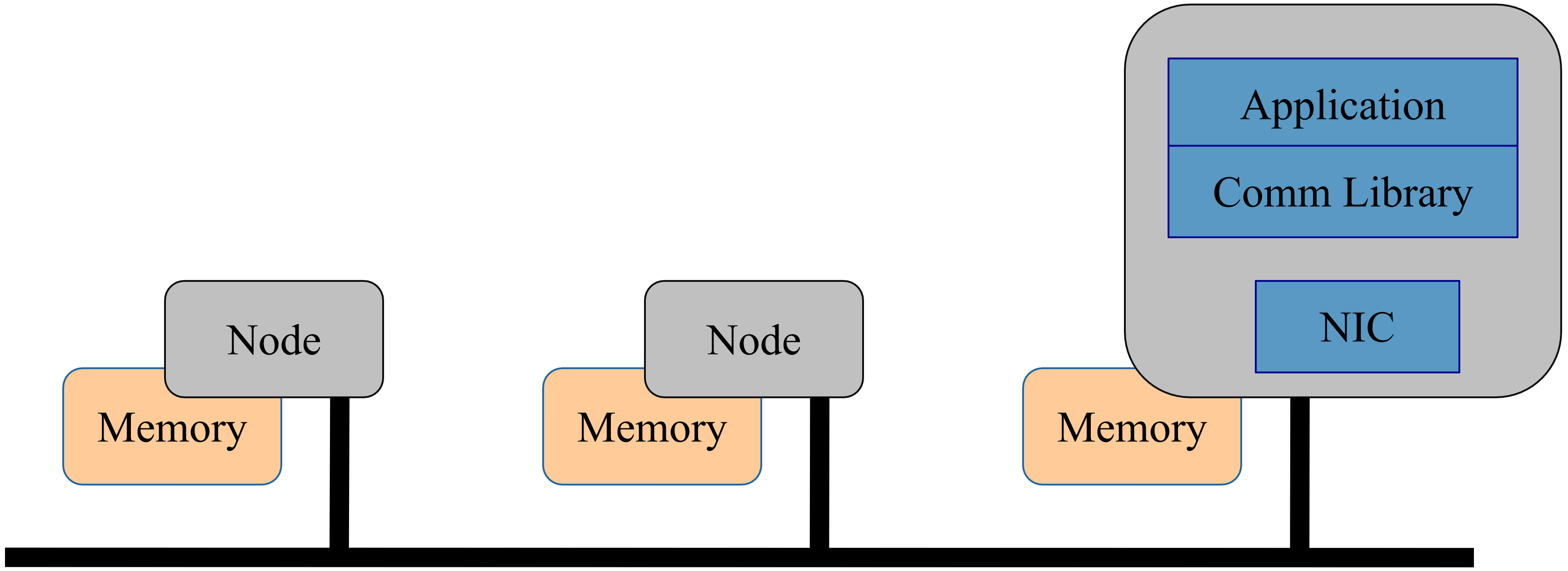
September 2024, IWOMP, Perth, Australia

Credit: DALL-E

The Message Passing Interface (MPI)

Designed in 1992, based on previous experiences with message passing libraries

- Based on the trend in the early 90ies towards shared memory architectures
- MPI 1.0 first ratified in 1994
- Started with simple point-to-point messaging and collectives
- Grew from there into broad functionality
- All documents at: http://www.mpi-forum.org/
- From the 25 year symposium: https://www.mcs.anl.gov/mpi-symposium/



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
- Several working groups

Key Contacts: WG Chairs and Forum Officers

Application Binary Interface (ABI)
- Jeff Hammond and Quincey Koziol

Collective Communication, Topology, Communicators, Groups
- Tony Skjellum

Fault Tolerance
- Aurélien Bouteiller and Ignacio Laguna

Fortran
- Jeff Hammond, Purushotham Bangalore and Tony Skjellum

HW Topologies
- Guillaume Mercier

Hybrid and Accelerator Programming
- Jim Dinan

I/O
- Quincey Koziol

Languages
- Martin Ruefenacht

Remote Memory Access
- Joseph Schuchart

Sessions
- Howard Pritchard

Tools
- Marc-Andre Hermanns

MPI Forum Officers
- Chair: Martin Schulz
- Secretary: Wesley Bland
- Treasurer: Brian Smith
- Editor: Bill Gropp

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
- Several working groups

Open membership

- Any organization is welcome to participate
- Individuals have to “associate” themselves with one organization
- 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
- Votes are typically intended to be “close to unanimous”

Forum Meetings

- Typically 4x per year – 2x virtual and 2x hybrid (one with EuroMPI)
- Informal weekly meeting slot on Wednesday (as needed)
- Working group meetings organized per group

The Brief History of MPI

Credit: DALL-E

First Version MPI 1.0/1.1

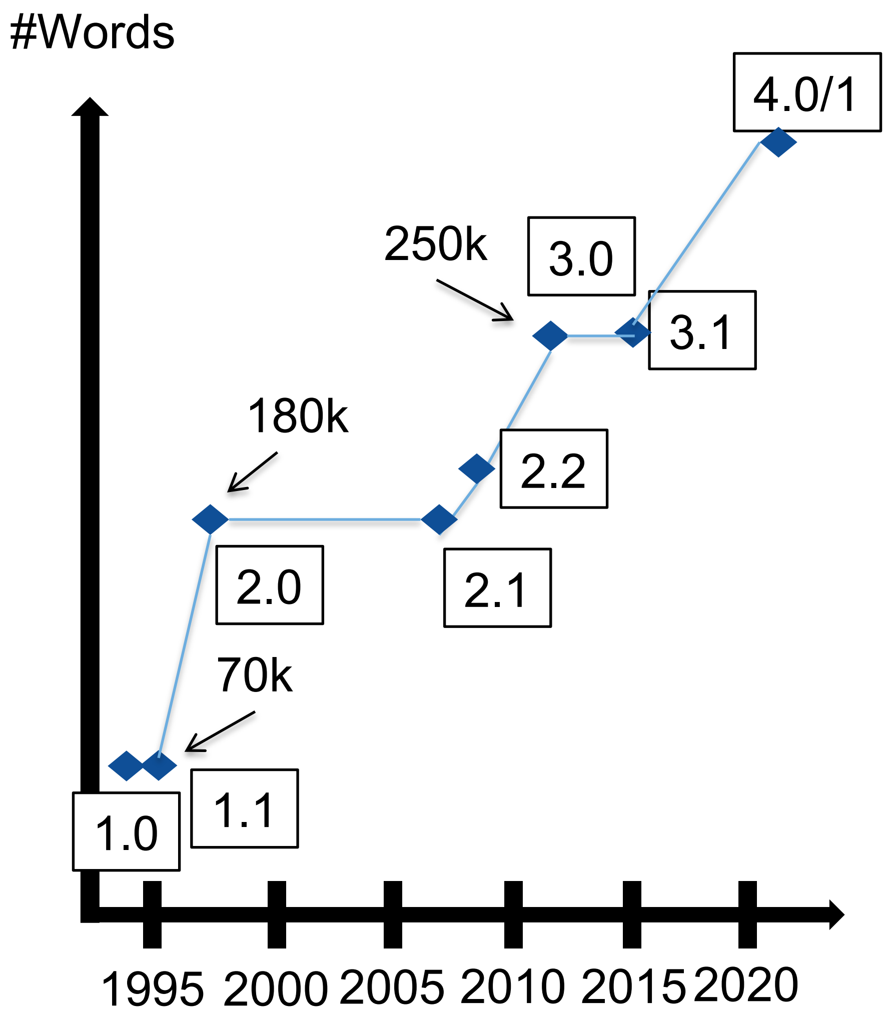
- Intended for the single core era
- Basic point-to-point (blocking, non-blocking & persistent)
- Blocking Collectives
- Basic datatypes
- Profiling interface

MPI 2.0/2.1/2.2

- Generalized concepts
- Added dynamic processes and Remote Memory Access
- Retro-fitted thread support
- More complex datatypes

MPI 3.0/3.1

- Nonblocking and Neighborhood collectives
- RMA and shared memory enhancements
- Fortran 2008 support
- First steps towards large counts
- New MPI tool interfaces



Where Are We?

MPI 4.0 was published in 2021

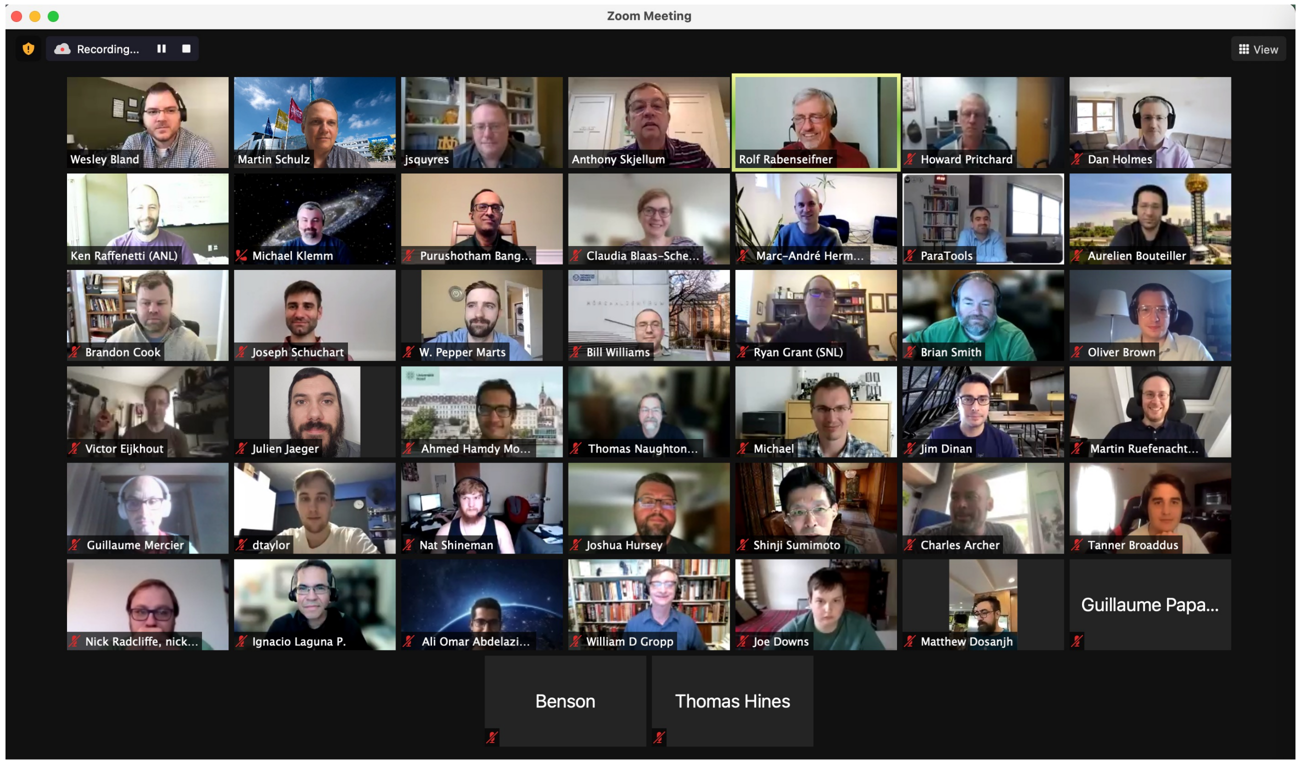
- Solution for “Big Count” operations
- **Partitioned Communication**
- Persistent Collectives
- Improved Error Handling
- **Topology Solutions**
- **New init options via MPI Sessions**
- And much more …

MPI 4.1 ratified on Nov. 2, 2023

- Major standard cleanup and clarifications
- **Memory kinds**

The work of the MPI Forum Continues

- MPI 4.2: support of an ABI
- Work on MPI 5.0 has begun targeting major new functionality

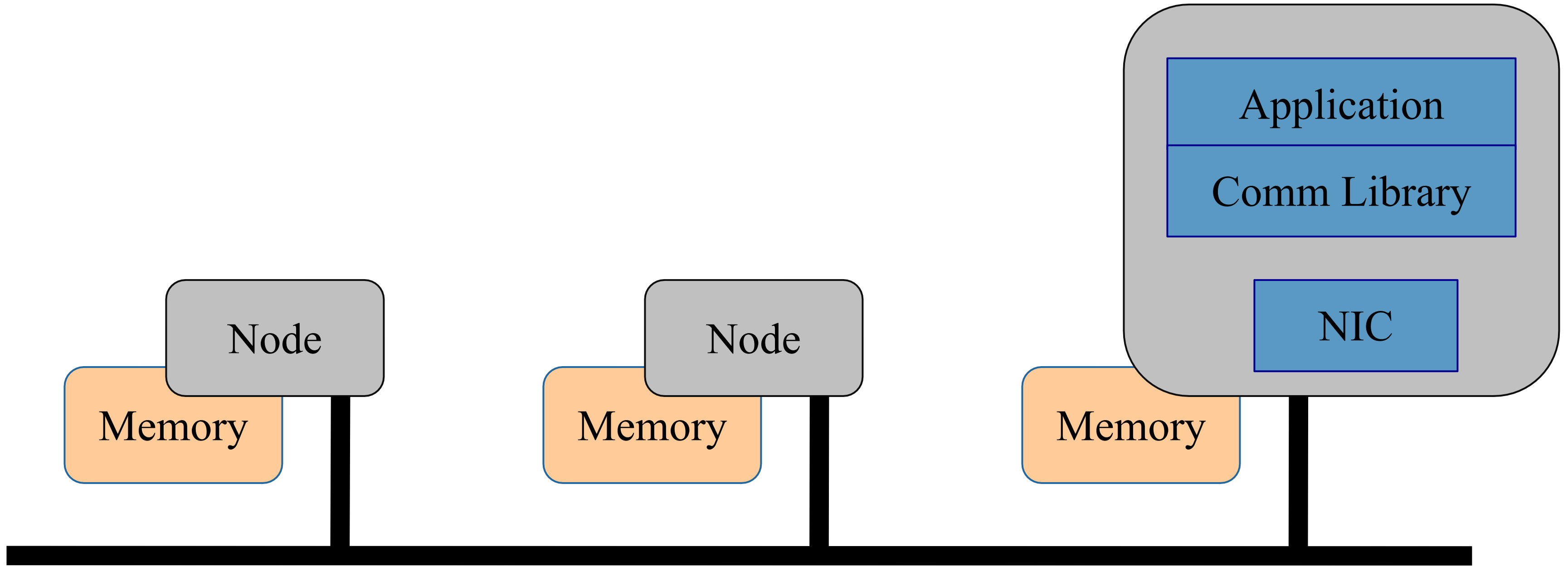


Join us:
www.mpi-forum.org

So, why a talk at IWOMP?

MPI was originaly designed for the single core era

- One MPI process per node
- No thread safety
- Weak progress model
- MPI runtime “runs the show”



So, why a talk at IWOMP?

MPI

+

X

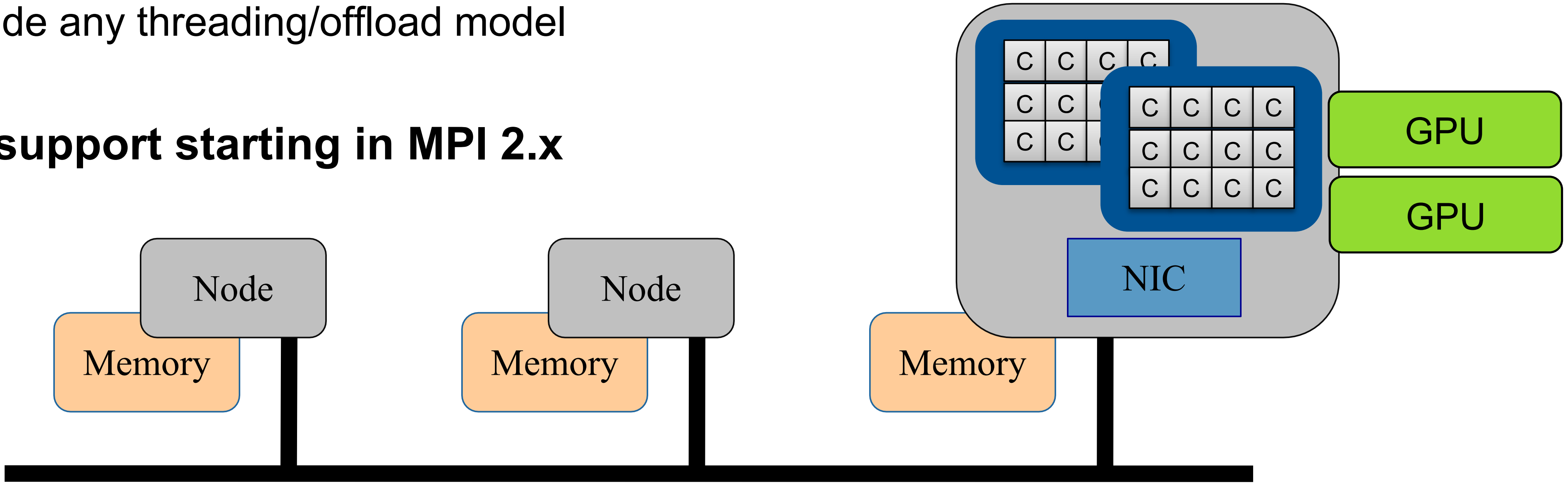
Today's systems look different

- Multi-core, Many core, Accelerators, ...
- Many threading models

Advantage of the MPI model

- Library/API approach
- Underlying program is a standard C/Fortran program
- Can include any threading/offload model

Threading support starting in MPI 2.x



MPI and OpenMP in Partnership

MPI + OpenMP is a common approach

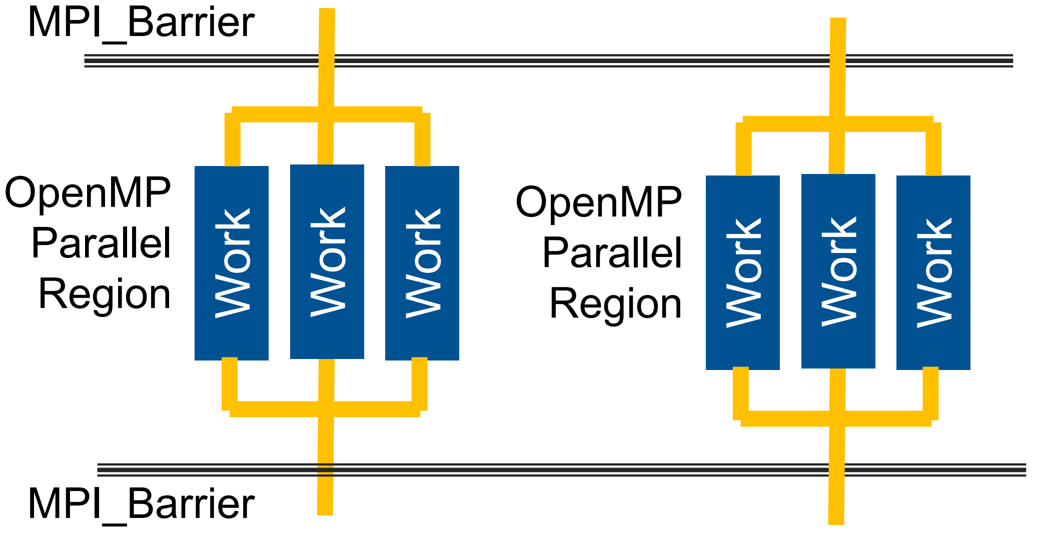
- OpenMP program(s) calling the MPI library
- MPI must be explicitly thread enabled
  - MPI_Init_thread
  - Pre-determined thread levels

Most common model

- Iterative codes
- MPI communication between iterations
- Iteration body parallelized with OpenMP

Many questions

- Ordering semantics in MPI
- Overhead of enabling threading
- Thread vs. process placement
- Memory management between runtimes
- etc.



Key Question:
How can MPI and OpenMP work (even) better together?

How much do runtimes have to interact?

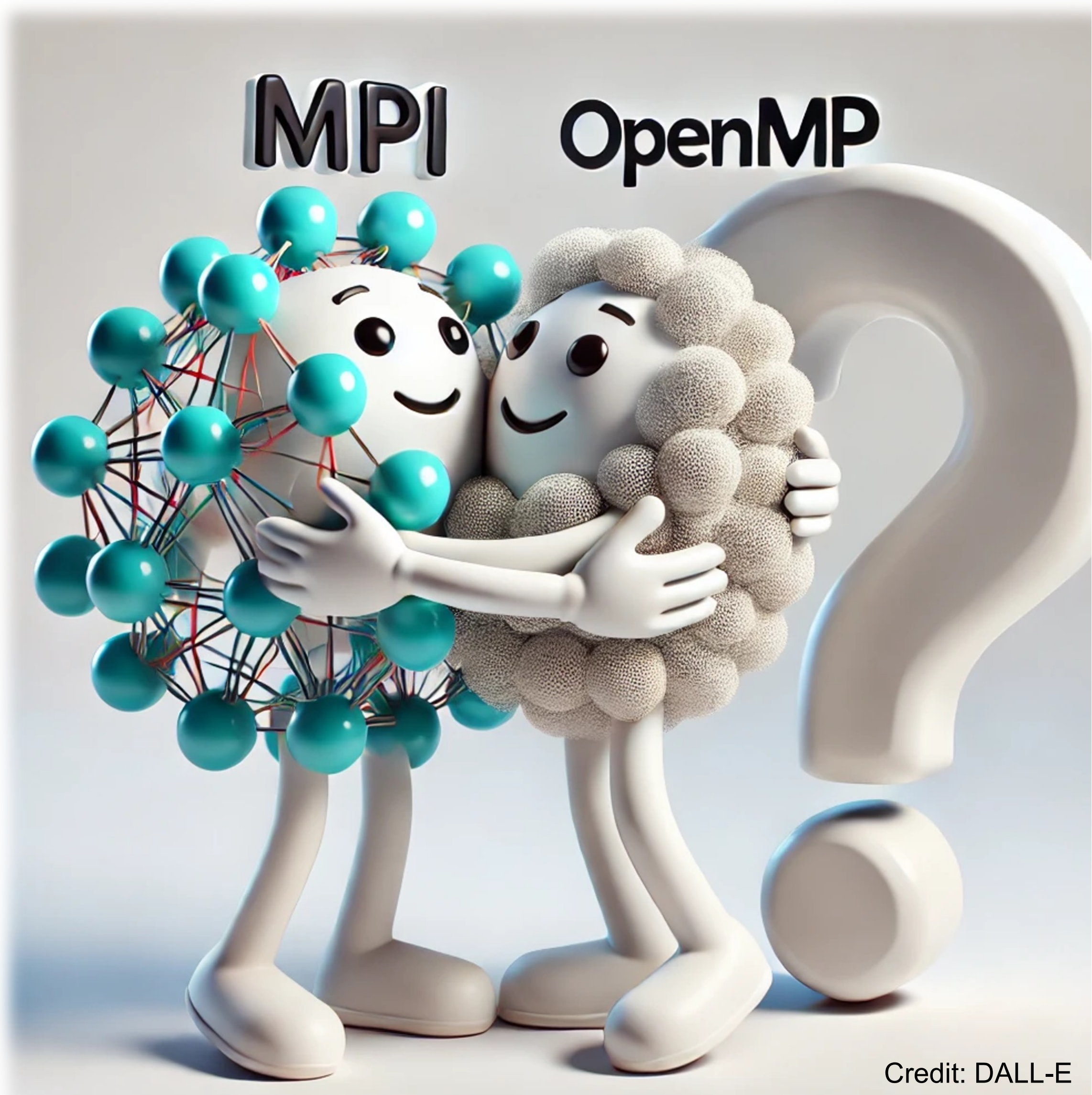
- How to capture each other's semantics?
- How do we do that without linking the standards?
- How do you ensure progress?
- How do you deal with mutual notifications?

How do you deal with shared resources

- Thread and process placement?
- Changing resources?

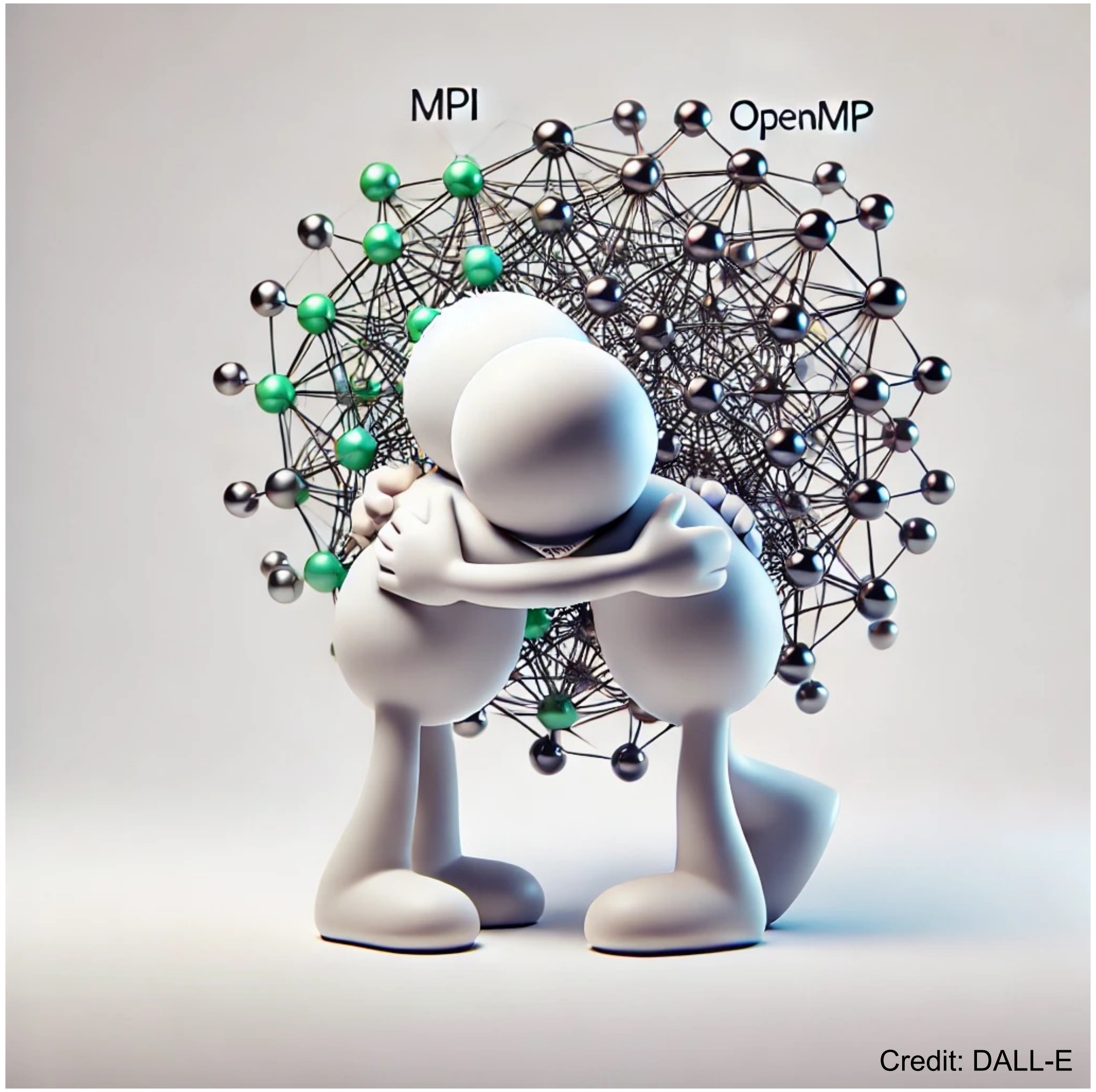
How to deal with memory allocations?

- Shared data structures?
- Interactions with accelerator memory?



Credit: DALL-E

Topics on Harmonizing MPI and OpenMP



Credit: DALL-E

Topic: General Threading

Current State in MPI and OpenMP

- MPI and OpenMP are fully independent
- MPI defines thread levels, which must be set
- OpenMP does not know anything about MPI runtime, especially blocking behavior

Challenges

- Identify the right thread levels
- Capture semantics
- Provide proper isolation where needed

Thread Levels in MPI

MPI has a special initialization routine for threaded behavior

```
int MPI_Init_thread(int *argc, char ***argv, int required, int *provided)
```

- Application states needed level of thread support
- MPI returns which one it supports

MPI_THREAD_SINGLE
- Only one thread exists in the application

MPI_THREAD_FUNNELED
- Multithreaded, but only the main thread makes MPI calls

MPI_THREAD_SERIALIZED
- Multithreaded, but only one thread at a time makes MPI calls

MPI_THREAD_MULTIPLE
- Multithreaded and any thread can make MPI calls at any time

Repeated Discussions

- Are these levels sufficient?
- How to contain overhead?
- Do the two middle thread levels help at all?
- Should MPI just be always thread-safe?
- What is their scope?

Introduction of MPI Sessions (MPI 4.0)

Attacking some fundamental problems in MPI

- MPI_COMM_WORLD is a very static resource
  - Minimized the complexity
- MPI_COMM_WORLD is immutable
  - Avoids many issues and reduces overhead (and made it not PVM ☺)
- MPI has no ability to isolate resources
  - This was not necessary in the past, very little to isolate
- MPI has no ability to “talk” to the runtime system
  - Explicit choice to improve portability, separation of concerns and matched HPC setups
- MPI is seen as not supporting new communities and industrial applications
  - HPC was the initial target and is still the main area

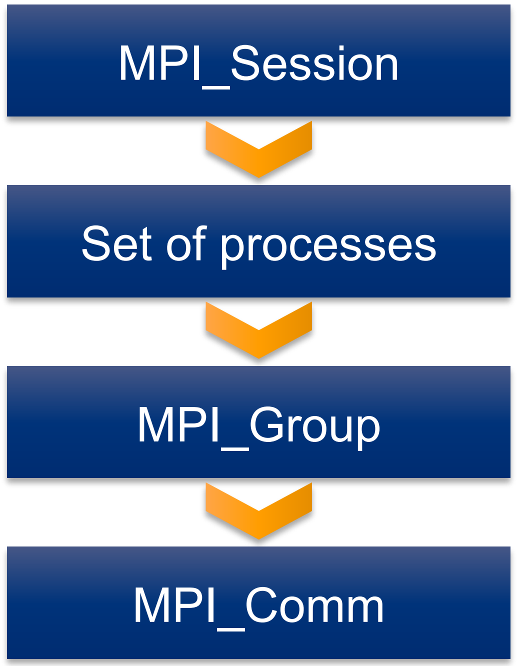
Question: How can we overcome this, without compromising MPI?

- Backwards compatible, but that is only part of it
- Same look and feel of MPI, maintain the learning curve
- Enable code reuse for existing codes

MPI Sessions

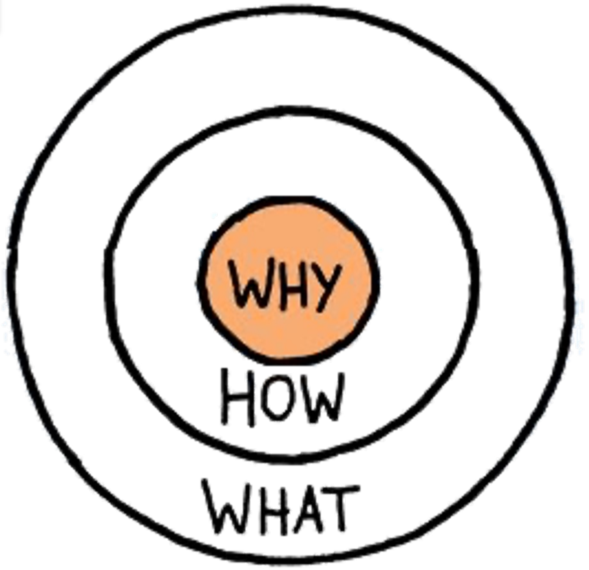
Instead of MPI_Init / MPI_COMM_WORLD:

1. **Get local access to the MPI library**
   Get a Session Handle
2. **Query the underlying run-time system**
   Get a “set” of processes
3. **Determine the processes you want**
   Create an MPI_Group
4. **Create a communicator with just those processes**
   Create an MPI_Comm



What does this do?

- Deliver runtime information of (changing) information to the MPI library
- Enables the ability to provide resource isolation between sessions
- Eliminate the need for a static resource MPI_COMM_WORLD



It’s a starting point!

Supported Thread Levels and Sessions

MPI_Init_session as Initializer for MPI

- Must/can contain thread level information
- Similar approach: using thread levels and request/provided interface

MPI Tools Information Interface

- Can be used before MPI_Init/MPI_Init_thread
- Can use internal threads as well
- Similar approach: using thread levels and request/provided interface

Different approaches must work together

- What happens when requests are different?
- First one returns thread levels and often anchors the thread level
  - Suboptimal solution
- Can this be made more flexible?
- Impact on OpenMP usage?

Ordering Issues with MPI_THREAD_MULTIPLE

MPI has ordering constraints

- E.g., when calling collectives
- Influences matching of MPI messages

Multiple threads can run in parallel and can call MPI routines

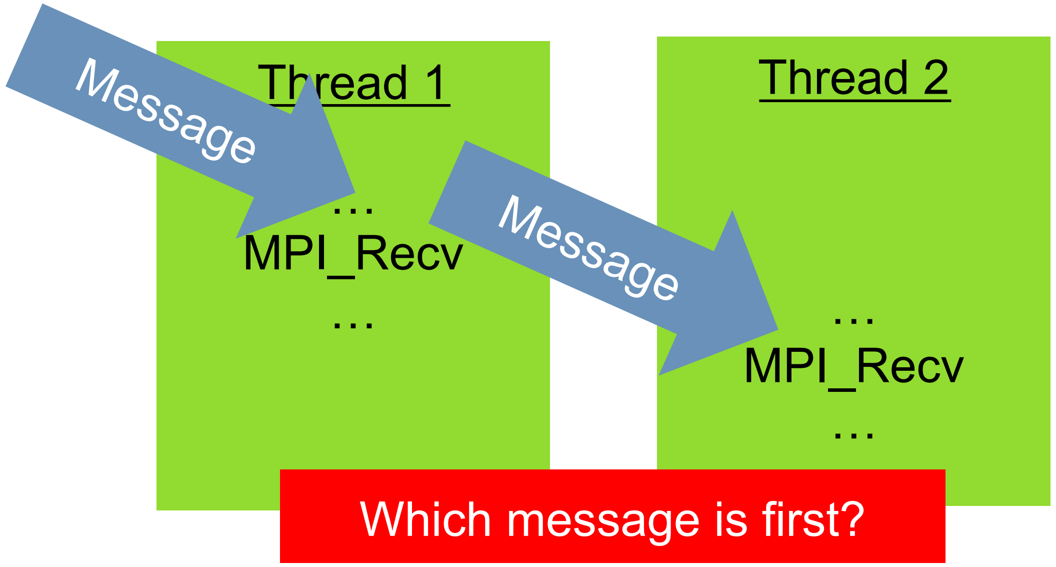
- Ordering rules have to be maintained in all cases
- User responsibility
- Semantics based on „entire MPI routines“
- Some threads are „locally concurrent“

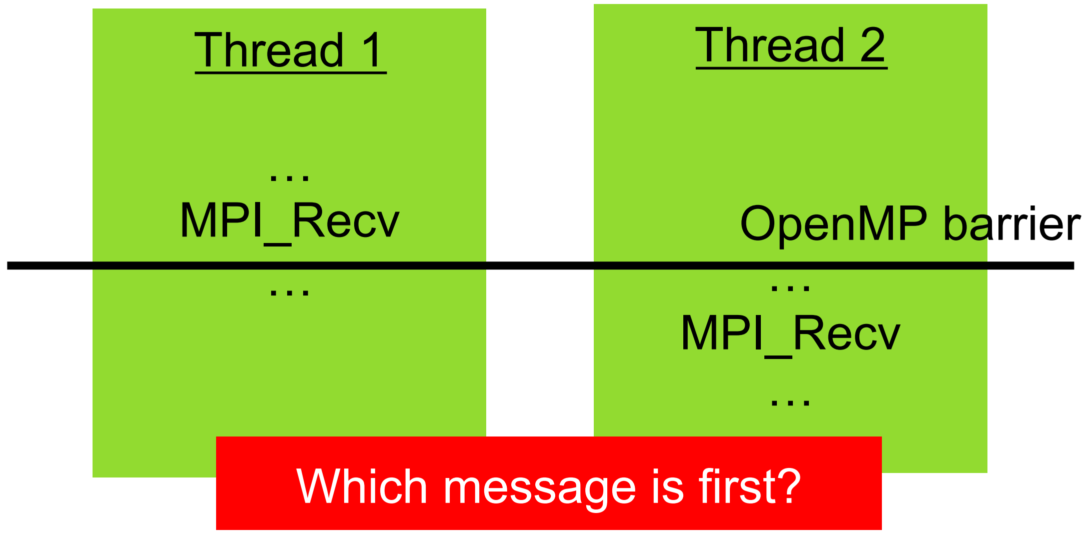
MPI-3.1 Section 3.5: If a process has a single thread of execution, then any two communications executed by this process are ordered. On the other hand, if the process is multithreaded, then the semantics of thread execution may not define a relative order between two send operations executed by two distinct threads. The operations are logically concurrent, even if one physically precedes the other. In such a case, the two messages sent can be received in any order. Similarly, if two receive operations that are logically concurrent receive two successively sent messages, then the two messages can match the two receives in either order.

Ordering Issues with MPI_THREAD_MULTIPLE

MPI-3.1 Section 3.5: If a process has a single thread of execution, then any two communications executed by this process are ordered. On the other hand, if the process is multithreaded, then the semantics of thread execution may not define a relative order between two send operations executed by two distinct threads. The operations are logically concurrent, even if one physically precedes the other. In such a case, the two messages sent can be received in any order. Similarly, if two receive operations that are logically concurrent receive two successively sent messages, then the two messages can match the two receives in either order.

What does logically concurrent mean?





"MPI+OpenMP" to "MPI in harmony with OpenMPI", Martin Schulz, IWOMP, September 2024

Topic: General Threading

Current State in MPI and OpenMP

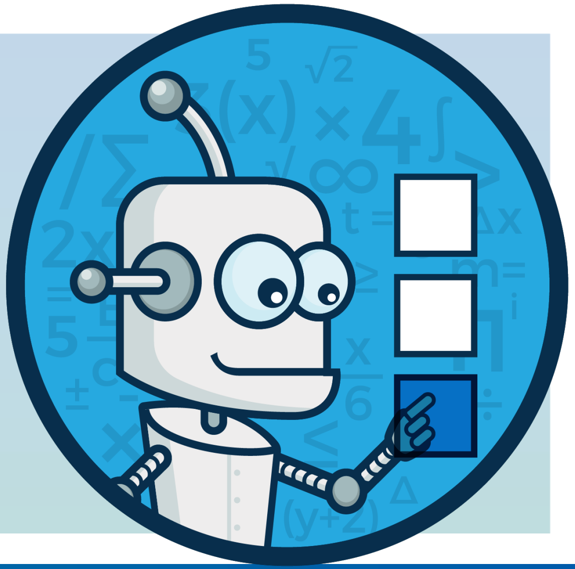
- MPI and OpenMP are fully independent
- MPI defines thread levels, which must be set
- OpenMP does not know anything about, especially blocking behavior

Challenges

- Identify the right thread levels
- Capture semantics
- Provide proper isolation where needed

Future directions / Steps / Questions

- Isolation of thread support across MPI sessions
  - Is there a role for the OpenMP runtime?
- Communication of synchronization between runtimes
  - Without having to assume the worst, i.e., full serialization?
  - Without explicitly knowing the other model



Topic: Continuations / MPI vs. Tasking

Current State in MPI and OpenMP

- MPI supports asynchronous operations with non-blocking operations
- OpenMP supports asynchronous tasks

Challenges

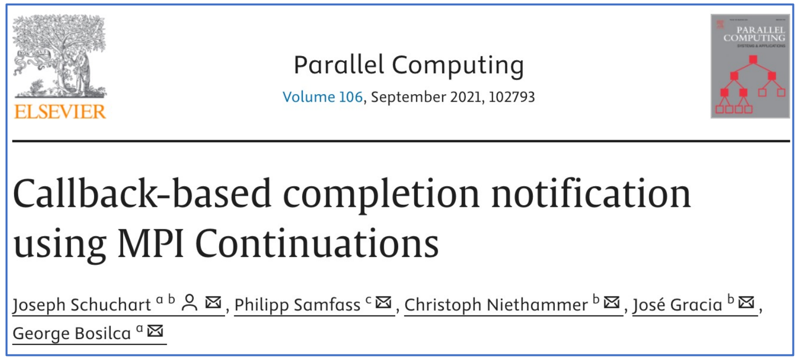
- No common notification
- Difficult to combine asynchronous approaches

From: Joseph Schuchart

Stony Brook University

MPI Continuations

- MPI relies on polling for completion of operations
- Proposal: add callbacks that trigger once operations are complete
- Attaching continuations to operations
- Registering continuations with a Continuation Request (CR)
- Error handling and propagation
- Status propagation
- Experiments with Pika, OpenMP, PaRSEC, C++ sender/receiver



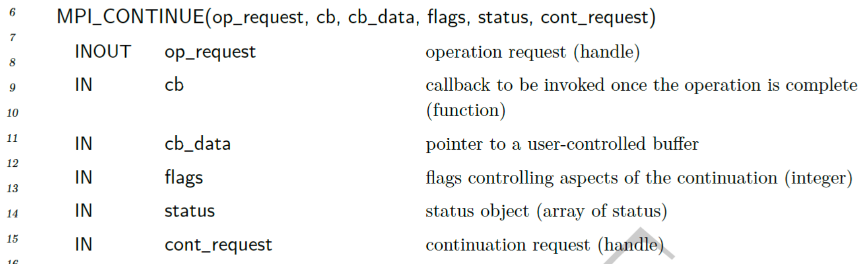
ELSEVIER

Parallel Computing

Volume 106, September 2021, 102793

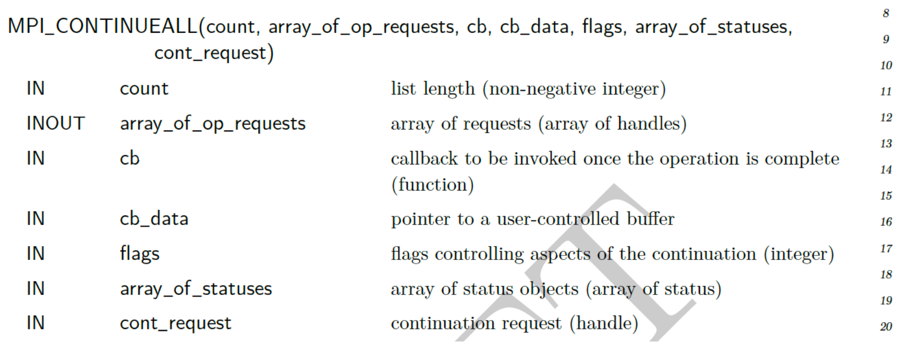
Callback-based completion notification using MPI Continuations

Joseph Schuchart [a b], Philipp Samfass [c], Christoph Niethammer [b], José Gracia [b], George Bosilca [a]



MPI_CONTINUE(op_request, cb, cb_data, flags, status, cont_request)

INOUT	op_request	operation request (handle)
IN	cb	callback to be invoked once the operation is complete (function)
IN	cb_data	pointer to a user-controlled buffer
IN	flags	flags controlling aspects of the continuation (integer)
IN	status	status object (array of status)
IN	cont_request	continuation request (handle)



MPI_CONTINUEALL(count, array_of_op_requests, cb, cb_data, flags, array_of_statuses, cont_request)

IN	count	list length (non-negative integer)
INOUT	array_of_op_requests	array of requests (array of handles)
IN	cb	callback to be invoked once the operation is complete (function)
IN	cb_data	pointer to a user-controlled buffer
IN	flags	flags controlling aspects of the continuation (integer)
IN	array_of_statuses	array of status objects (array of status)
IN	cont_request	continuation request (handle)

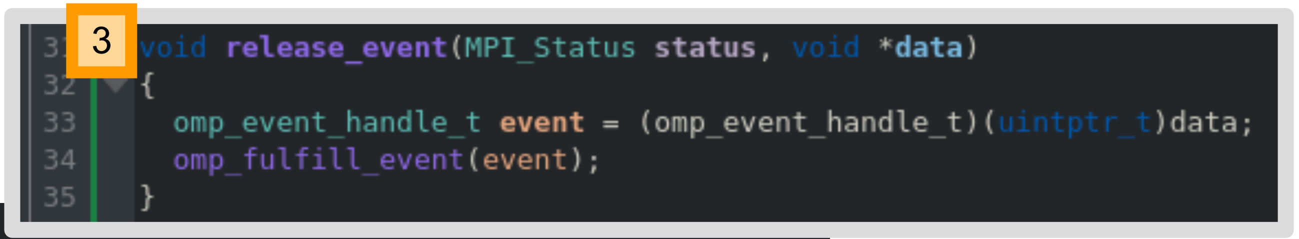
FAR BEYOND

https://github.com/mpiwg-hybrid/hybrid-issues/issues/6

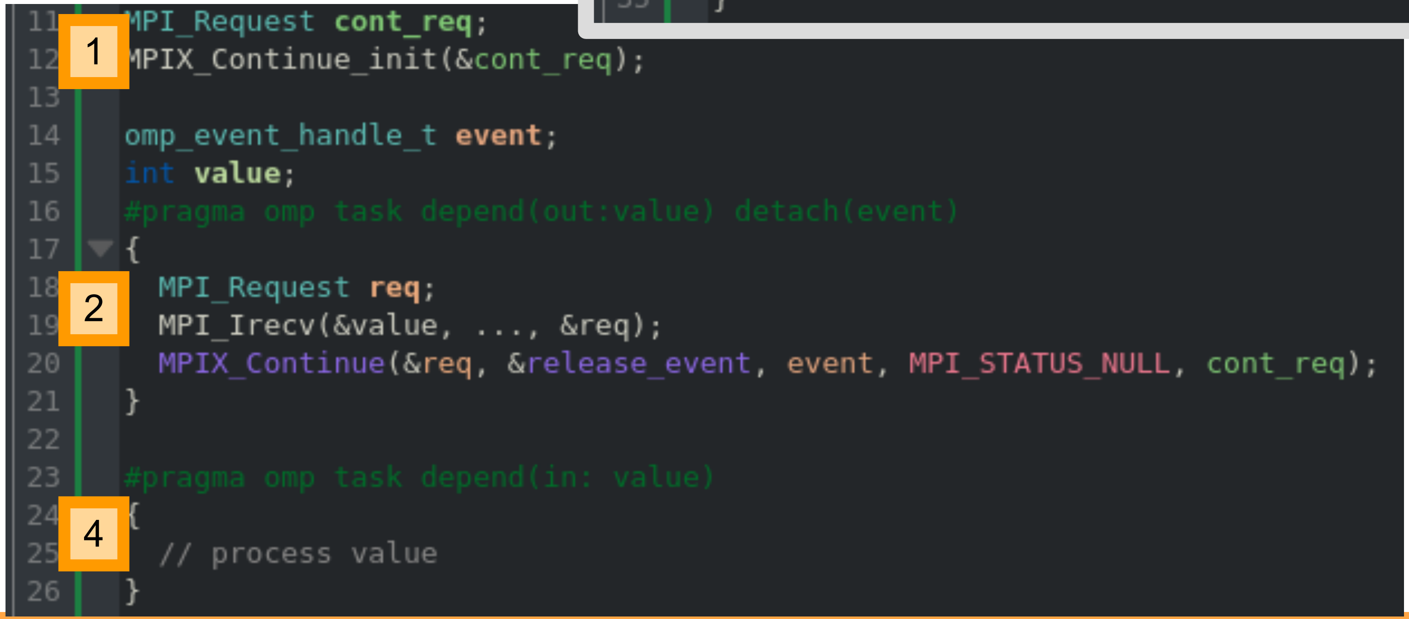
iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

Proposal for Thread Continuations

Idea: **Treat the completion of an MPI operation as continuation of some activity**
Ability to couple with OpenMP events and dependencies



```c
void release_event(MPI_Status status, void *data)
{
  omp_event_handle_t event = (omp_event_handle_t)(uintptr_t)data;
  omp_fulfill_event(event);
}
```



```c
MPI_Request cont_req;
MPIX_Continue_init(&cont_req);

omp_event_handle_t event;
int value;
#pragma omp task depend(out:value) detach(event)
{
  MPI_Request req;
  MPI_Irecv(&value, ..., &req);
  MPIX_Continue(&req, &release_event, event, MPI_STATUS_NULL, cont_req);
}

#pragma omp task depend(in: value)
{
  // process value
}
```

"Callback-based completion notification using MPI Continuations,"
Joseph Schuchart, Christoph Niethammer, José Gracia, George Bosilca, Parallel Computing, 2021.

"MPI Detach - Asynchronous Local Completion,"
Joachim Protze, Marc-André Hermanns, Ali Demiralp, Matthias S. Müller, Torsten Kuhlen. EuroMPI '20.

Topic: Continuations / MPI vs. Tasking

Current State in MPI and OpenMP

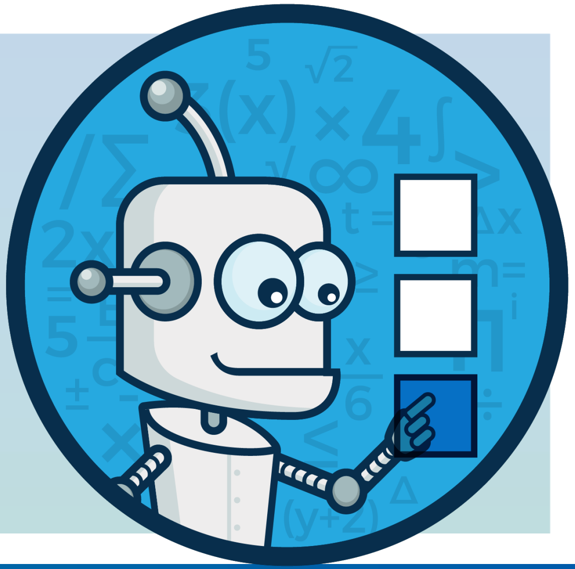
- MPI supports asynchronous operations with non-blocking operations
- OpenMP supports asynchronous tasks

Challenges

- No common notification
- Difficult to combine asynchronous approaches

Future directions / Steps / Questions

- Treat the completion of an MPI operation as continuation of some activity
- Ability to couple with OpenMP events and dependencies
- Needs interactions between runtimes via defined API
- Can this be API agnostic from an MPI point of view in any case?



Topic: Topologies

Current State in MPI and OpenMP

- MPI supports automatic topology optimization left up to MPI implementations
  - Typically not well optimized
- Newer versions allow for adaptive queries
  - No predefinitions
- OpenMP has places concept and has predefined names for some items

Challenges

- Two runtimes working on this at the same time and may conflict
- Naming of items not synchronized
- Both assume static topologies

New Ways to Adapt to Topologies

New systems are very hierarchical

- On node and whole system
- Application mapping is critical
- Need topology-aware communicators

New Ways to Adapt to Topologies

New systems are very hierarchical

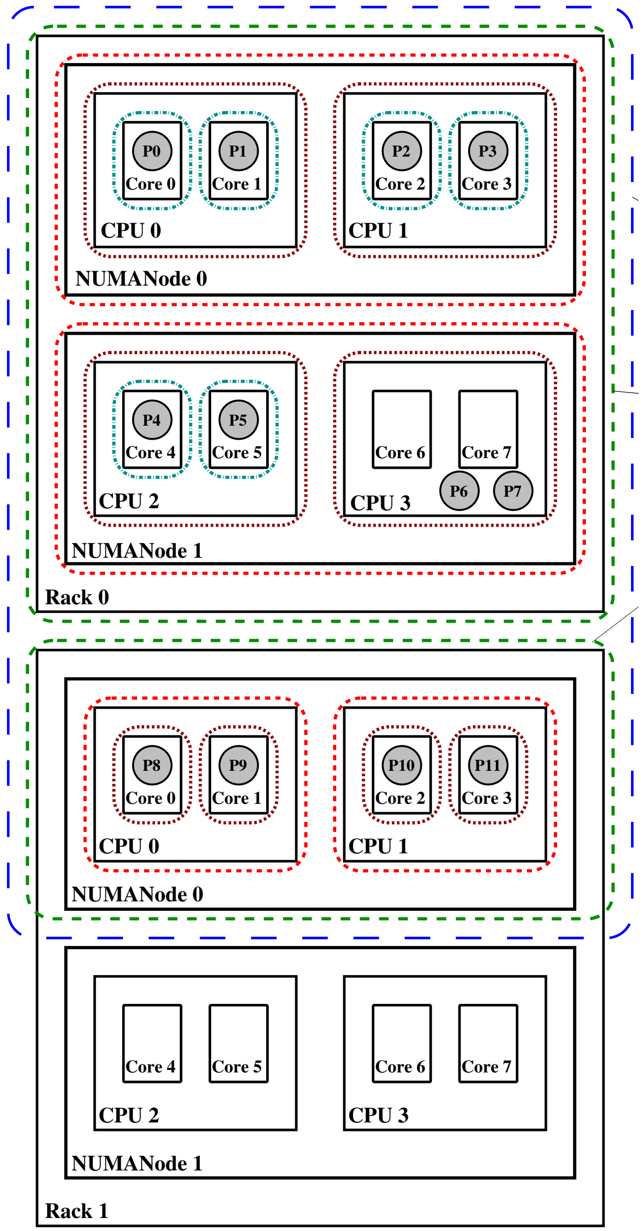
- On node and whole system
- Application mapping is critical
- Need topology-aware communicators

Guided Mode

- MPI_COMM_SPLIT_TYPE
- Special split type with info key to specify level
- No predefine names, but can query

Unguided Mode

- Start at MPI_COMM_WORLD
- Step-wise go to lower levels until leaf is reached

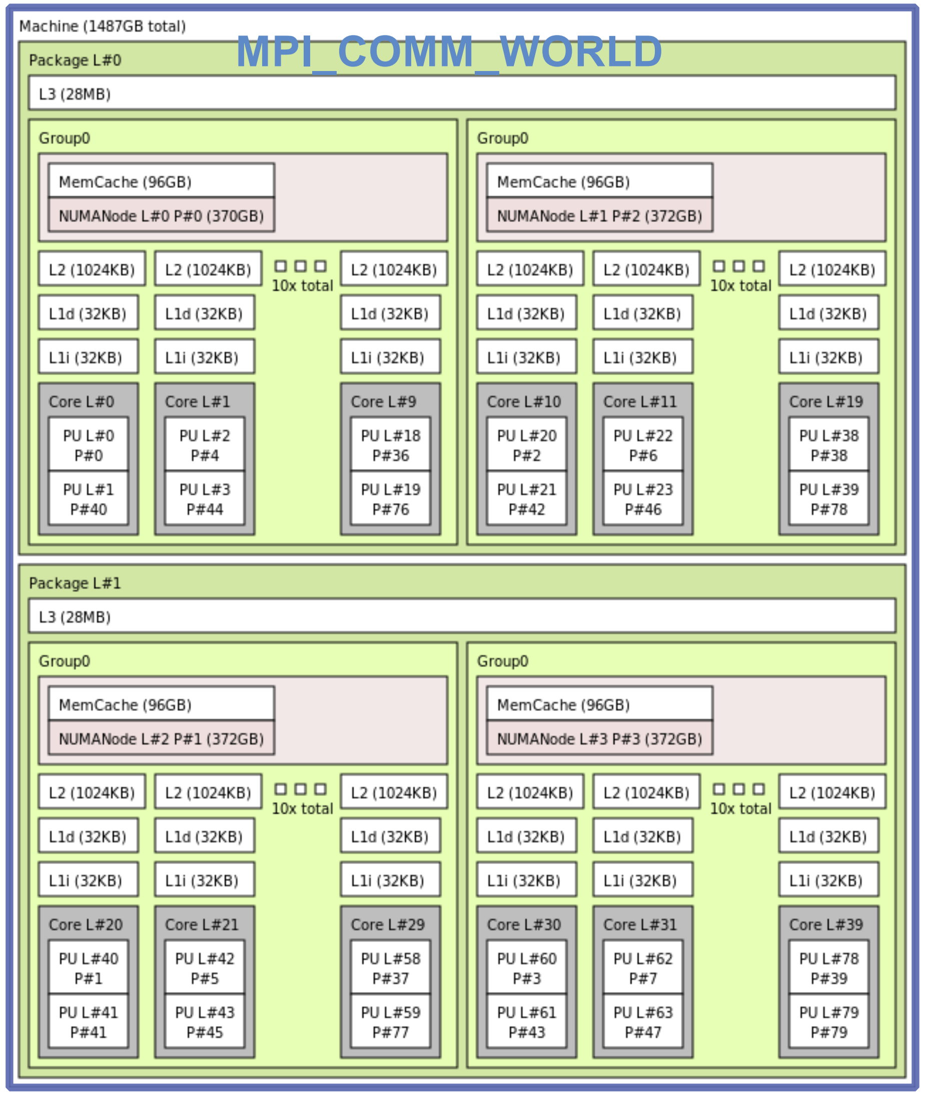


Graphics/Example from Guillaume Mercier, INRIA

Example

MPI_Init(…);

MPI_Comm_rank(MPI_COMM_WOLRD, &rank);



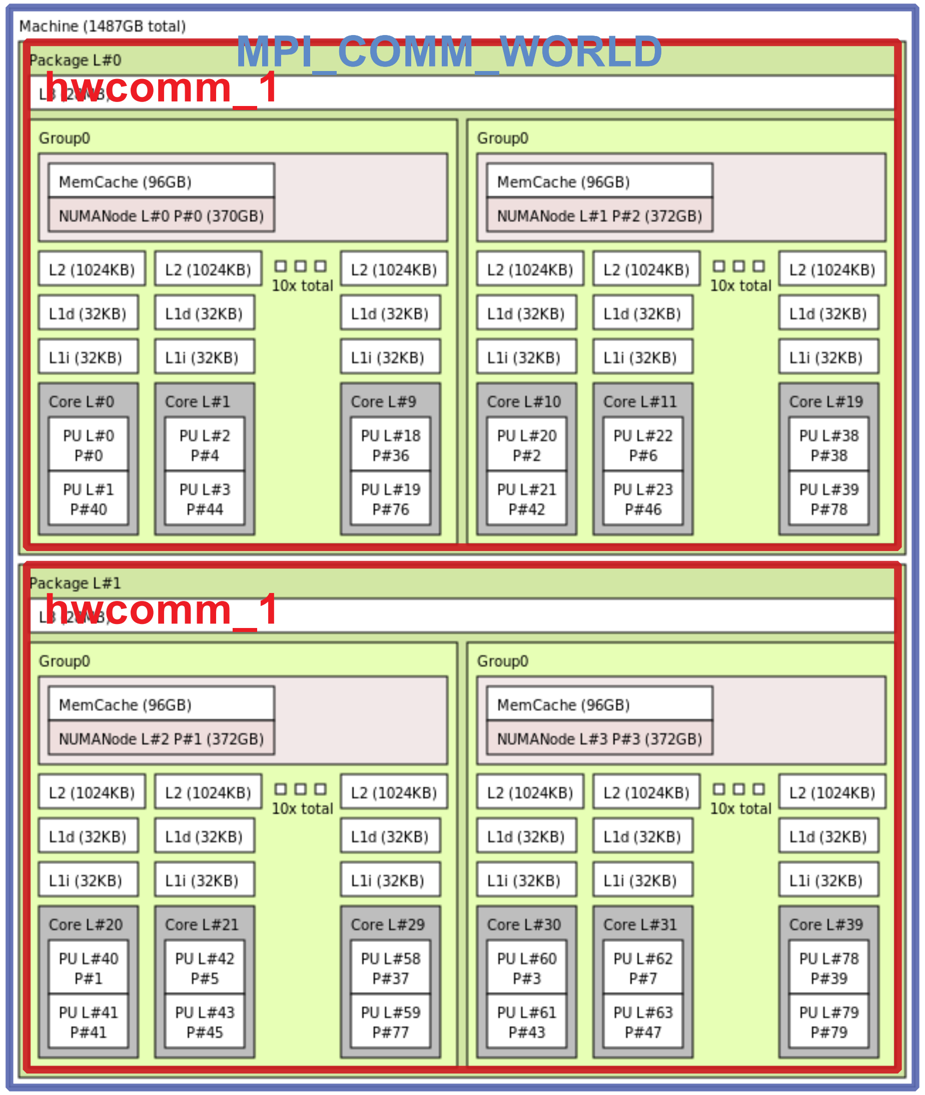
Graphics/Example from Guillaume Mercier, INRIA

Example

MPI_Init(…);

MPI_Comm_rank(MPI_COMM_WOLRD, &rank);

MPI_Comm_split_type(MPI_COMM_WORLD,
MPI_COMM_TYPE_HW_UNGUIDED,
rank,info,&hwcomm_1);



Graphics/Example from Guillaume Mercier, INRIA

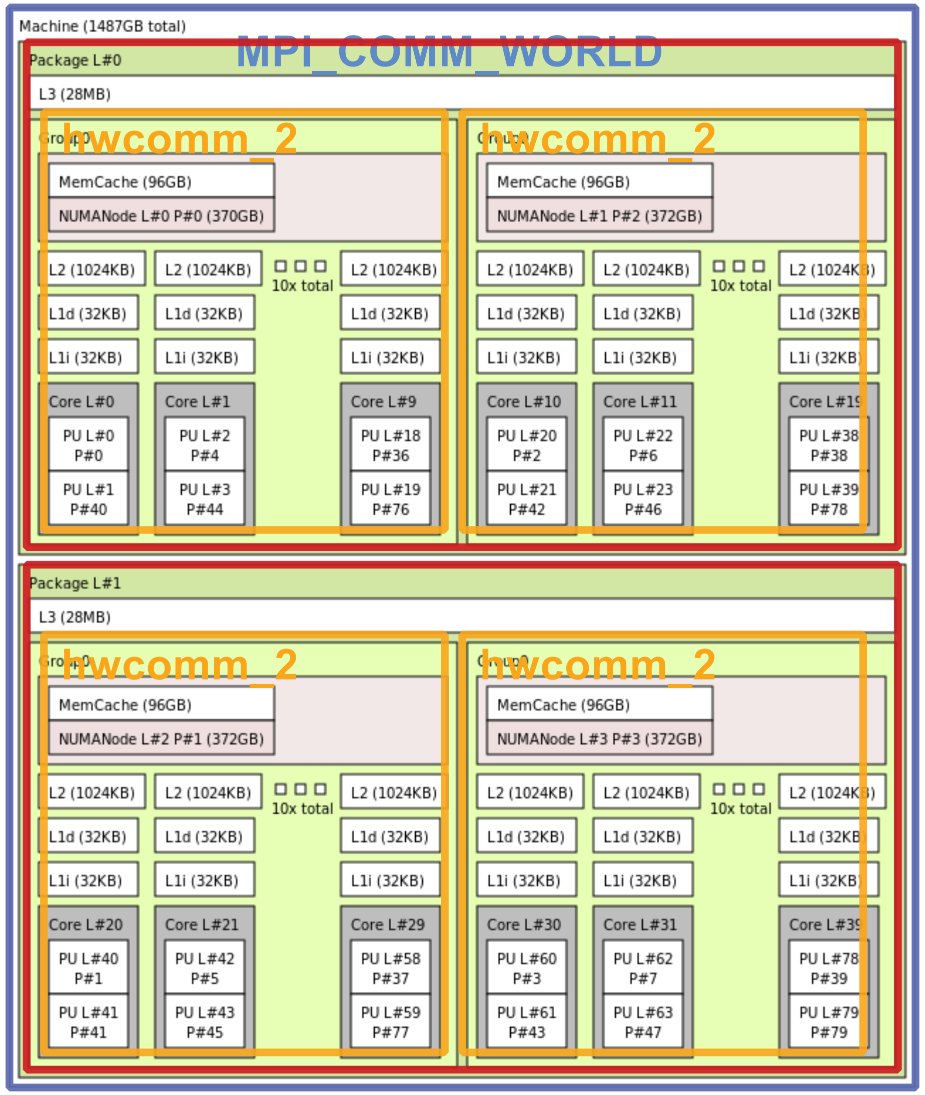
Example

```
MPI_Init(…);

MPI_Comm_rank(MPI_COMM_WOLRD, &rank);

MPI_Comm_split_type(MPI_COMM_WORLD,
          MPI_COMM_TYPE_HW_UNGUIDED,
          rank,info,&hwcomm_1);

MPI_Comm_split_type(hwcomm_1,
          MPI_COMM_TYPE_HW_UNGUIDED,
          rank,info,&hwcomm_2);
```



Graphics/Example from Guillaume Mercier, INRIA

Adding Session-based Malleability Concepts

MPI Sessions provide the ability to reason about malleability

- Dynamic addition/reduction of processes
- Could change the resource utilization on a node

Approach 1: Objects form a natural group: an "MPI bubble"

- Isolated from the world in terms of communication
- Could be revoked/popped without global impact

  → Global approach, collective across processes

Approach 2: Exploiting process groups

- One process manages resource interactions
- Process sets and their URIs offer a good option for this
- See also Huber et al. @ EuroMPI 2022

Towards Dynamic Resource Management with MPI Sessions and PMIx
Huber et al., EuroMPI 2022

In both approaches:

- Impact on thread resources on node

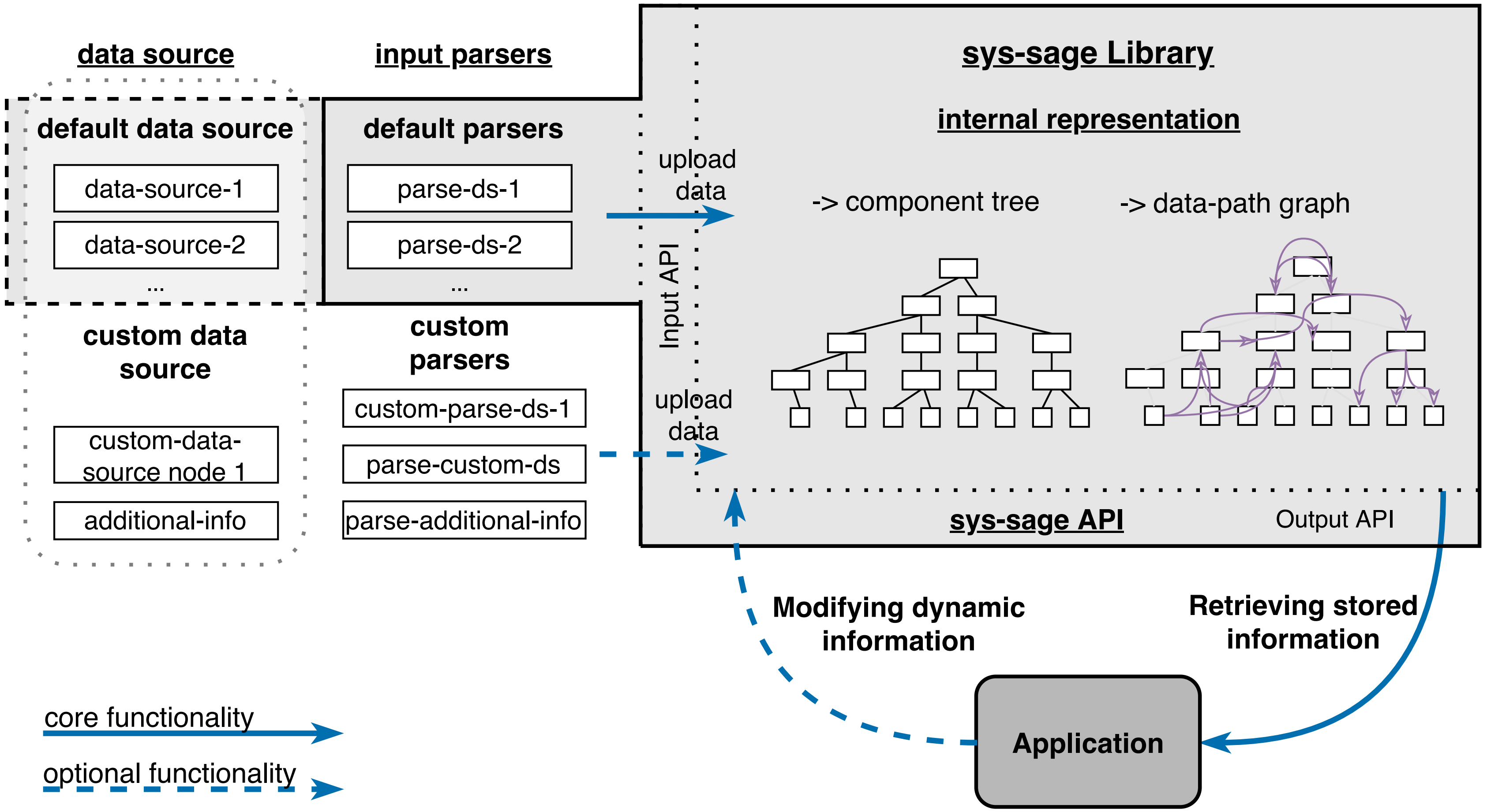
Topology Detection with *sys-sage*

Need to capture topology

- Dynamic
- With attributes
- Expandable

New project: *sys-sage*

- Capture topology
- Dual representation
  - Component tree
  - Data-path graph
- Express dynamic behavior
- Capture system changes



A Unified Representation of Dynamic Topologies & Attributes on HPC Systems, Stepan Vanecek, Martin Schulz, ICS 2024

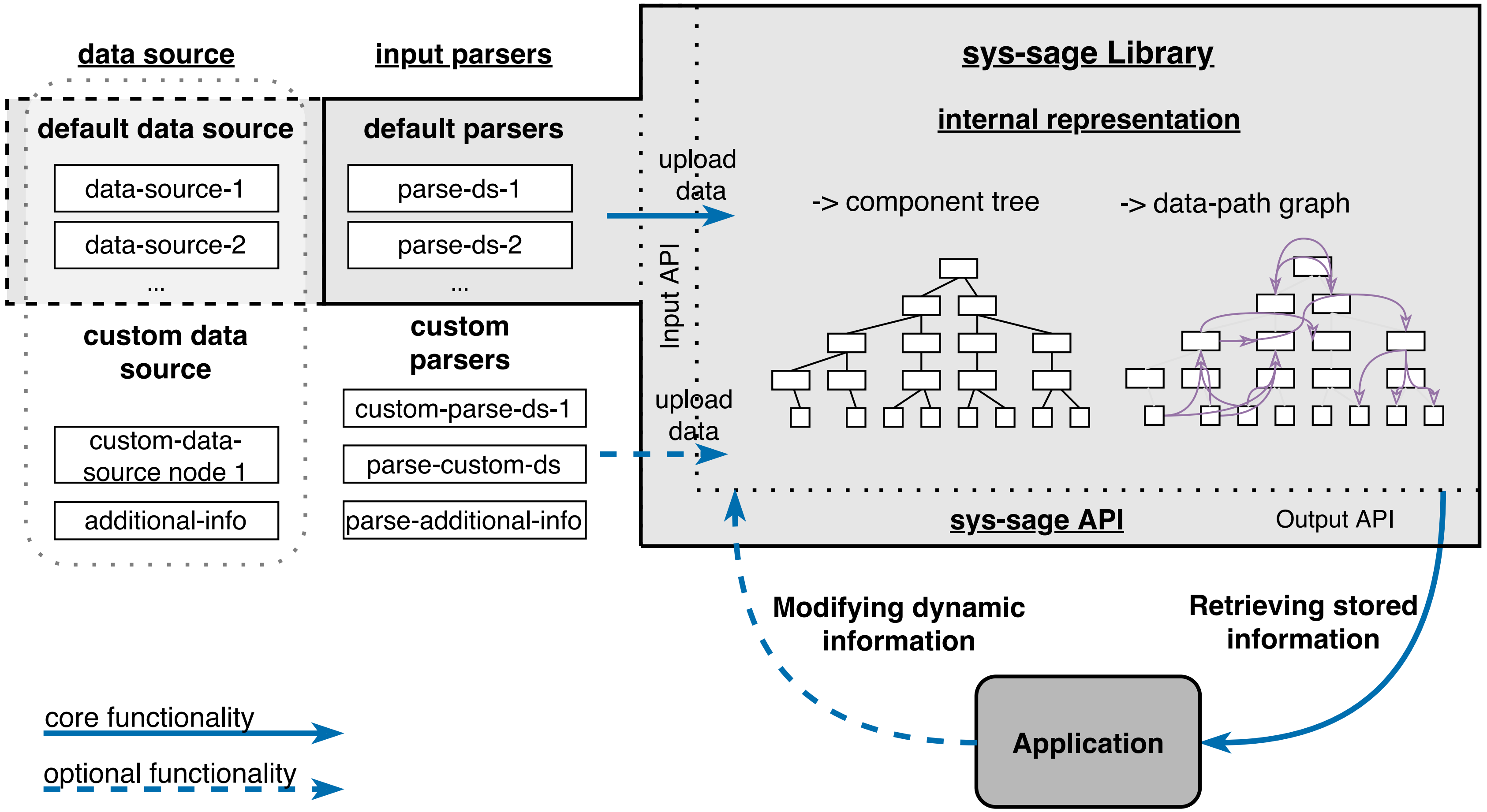
Topology Detection with *sys-sage*

Need to capture topology

- Dynamic
- With attributes
- Expandable

New project: *sys-sage*

- Capture topology
- Dual representation
  - Component tree
  - Data-path graph
- Express dynamic behavior
- Capture system changes



Currently single node focused

- Adaptation to distributed environments in progress
- Usage of MPI for implementation, data source and data target

Topic: Topologies

Current State in MPI and OpenMP

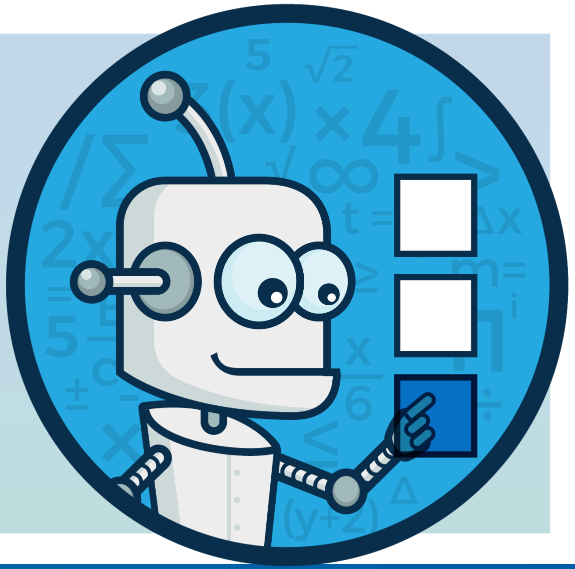
- MPI supports automatic topology optimization left up to MPI implementations
  - Typically not well optimized
- Newer versions allow for adaptive queries
  - No predefinitions
- OpenMP has places concept and has predefined names for some items

Challenges

- Two runtimes working on this at the same time and may conflict
- Naming of items not synchronized
- Both assume static topologies

Future directions / Steps / Questions

- Homogenization of resource names
- Mapping of places to topologies
- Enable cooperation between runtimes behind the scenes
  - Ideally no new public APIs, need to hide
  - Interaction with resource manager → PMI(x)



Topic: Lightweight Communication

Current State in MPI and OpenMP

- Communication independent of threading and tasking
- Each thread/task must call heavy weight MPI call

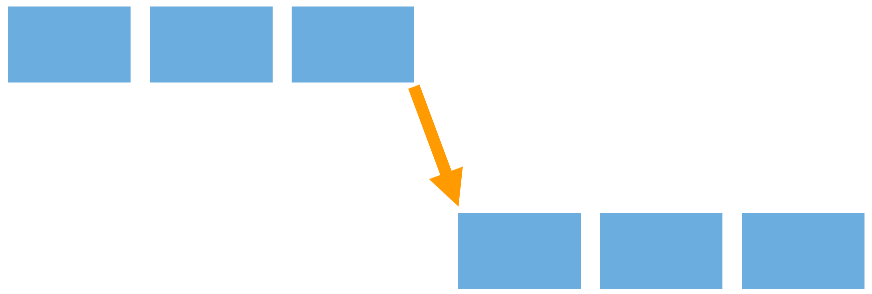
Challenges

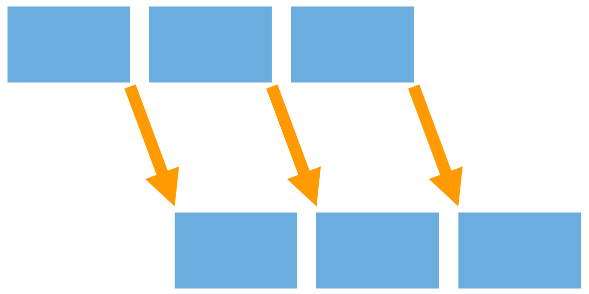
- Communication patterns often coordinated between threads (parallel execution of one problem)
- Leaves coordination to user
- Forces strict thread assignments, even if not needed

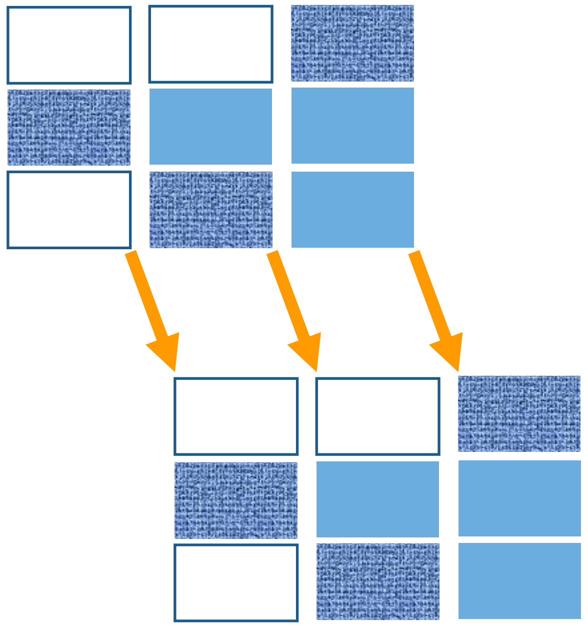
Partitioned Communication (MPI 4.0)

Core idea – efficient highly concurrent communication

- Built on the concept of persistent P2P communication
- Send and receive buffers are split into (possibly different) partitions
  - Fill each partition and mark it as ready
  - Individual notifications for each arriving partition







Notifications - on send and receive side – are light-weight

- May be driven from light weight environments, without entire MPI stack
- May need additional synchronization to trigger message transfer safely

Partitioned Communication for Thread Support

```
MPI_Psend_init(..., &request);
for (...) {
  MPI_Start(&request);
  #pragma omp parallel
  {
            kernel(..., request);
  }
  MPI_Wait(&request);
}
MPI_Request_free(&request);
```

```
Thread:

kernel(..., MPI_Request request)
{
  int i = my_partition[my_id];
  /* Compute and fill partition i then mark ready: */
  MPI_Pready(i, request);
}
```

Heavy weight MPI communication outside of parallel region

- Inside only light-weight triggering (easier thread safety)
- Partition for each thread, but single communication
- Each thread signals when it is ready
- MPI can optimize for latency or bandwidth (or shift)
- Should OpenMP know about these routines for optimization purposes?

Partitioned Communication for Thread Support

```
MPI_Psend_init(..., &request);
for (...) {
  MPI_Start(&request);
  #pragma omp parallel
  {
            kernel(..., request);
  }
  MPI_Wait(&request);
}
MPI_Request_free(&request);
```

```
Thread:

kernel(..., MPI_Request request)
{
  int i = my_partition[my_id];
  /* Compute and fill partition i then mark ready: */
  MPI_Pready(i, request);
}
```

Extensions to basic concept

- Guarantees for readiness on the receiver
- Collective versions
- Different send type options?
- Non persistent version?

Partitioned Communication for Thread Support

```
MPI_Psend_init(..., &request);
for (...) {
  MPI_Start(&request);
  #pragma omp target
  {
          GPU_kernel(..., request);
  }
  MPI_Wait(&request);
}
MPI_Request_free(&request);
```

```
Accelerator:

kernel(..., MPI_GPU_Request request)
{
  int i = my_partition[my_id];
  /* Compute and fill partition i then mark ready: */
  MPI_Pready(i, request);
}
```

Extensions to basic concept

- Guarantees for readiness on the receiver
- Collective versions
- Different send type options?
- Non persistent version?

Extensions to accelerators

- Heavy weight MPI on host process
- Light-weight triggers on accelerators (without OS)
- Support for thread/warp/block (or similar) variants
- Requires translation of requests
- Does OpenMP need to know of the semantics?

Requires GPU bindings in target regions for MPI

Topic: Lightweight Communication

Current State in MPI and OpenMP

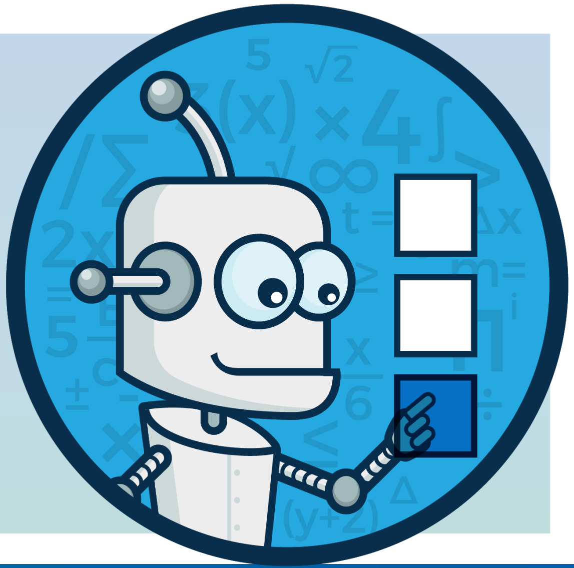
- Communication independent of threading and tasking
- Each thread/task must call heavy weight MPI call

Challenges

- Communication patterns often coordinated between threads (parallel execution of one problem)
- Leaves coordination to user
- Forces strict thread assignments, even if not needed

Future directions / Steps / Questions

- Explicit preparation of the receive buffer to enable direct transfers
- Would a deeper runtime integration be helpful to coordinate
- Number of partitions, thread to partition mapping, …?
- How should collectives be optimized?



Topic: Memory Allocation Types

Current State in MPI and OpenMP

- MPI add memory allocation kinds in MPI 4.1 to support device memories
- Support for different vendors can be queried and then explicitly requested
- OpenMP is fully vendor agnostic

Challenges

- Ideally we need an OMP memory kind
- MPI cannot query the supported type of the device, though
- MPI cannot use OMP maintained memory

Allocator Kind Info (MPI 4.1)

Use MPI info to provide users with a portable solution to:

1. Detect whether accelerator memory is supported by the MPI library
2. Request support for accelerator memory from the MPI library (when using Sessions)
3. Constrain usage of accelerator memory to specific communicators, windows, etc.

mpi_request_memory_alloc_kind

- Request support for memory allocator kind from the MPI library

mpi_assert_memory_alloc_kind

- Assert memory kinds used by the application on the given MPI object

mpi_memory_alloc_kind

- Memory kinds supported by the MPI library

Slide by Jim Dinan, NVIDIA

Request Support for CUDA Allocated Memory

```
bool cuda_aware = false;
int len = MPI_MAX_INFO_VAL, flag = 0;
char *val = malloc(MPI_MAX_INFO_VAL);
MPI_Info info;

MPI_Info_create(&info);
MPI_Info_set(info, “mpi_memory_alloc_kinds”, “cuda:device”);
MPI_Session_init(info, MPI_ERRORS_ARE_FATAL, &session);
MPI_Info_free(&info);

MPI_Session_get_info(session, &info);
MPI_Info_get_string(info, “mpi_memory_alloc_kinds”, &len, val, &flag);

// Check mpi_memory_alloc_kind for “cuda:device”
while (flag && (kind = strsep(&val, ",")) != NULL) {
  if (strcasecmp(kind, "cuda:device") == 0) {
    cuda_aware = true;
    break;
  }
}
```

Slides by Jim Dinan, NVIDIA

Status and Challenges with OpenMP

Currently adding a side document to define memory kinds

- Separate standard and vendor specific extensions
- Ability to quickly adjust, between standard versions
- Currently support for the three major GPU vendors

Only works on memory used in the underlying native model

- MPI needs to know the exact type of memory and how to access it
- Generic OMP memory kind has been proposed
  - Would be useful
  - BUT: MPI cannot know the underlying vendor to trigger access

Would need OpenMP extensions

- Enable per device controlled memory operation
  - Intel extensions available in their compiler
- Ability to query memory kinds from within OpenMP

Topic: Memory Allocation Types

Current State in MPI and OpenMP

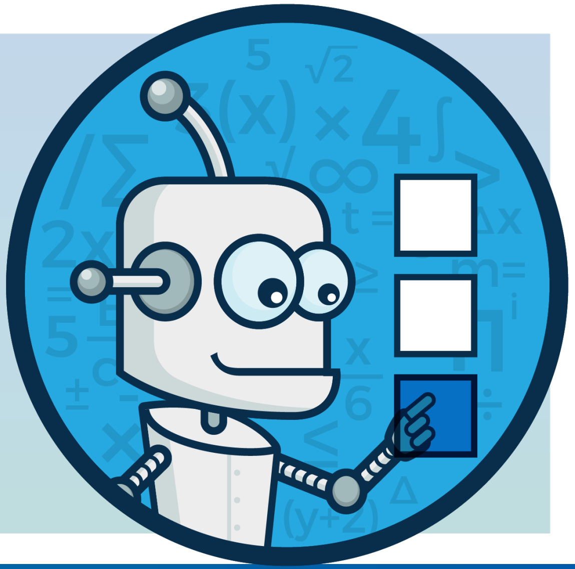
- MPI add memory allocation kinds in MPI 4.1 to support device memories
- Support for different vendors can be queried and then explicitly requested
- OpenMP is fully vendor agnostic

Challenges

- Ideally we need an OMP memory kind
- MPI cannot query the supported type of the device, though
- MPI cannot use OMP maintained memory
- Limits MPI's ability to optimize communication from/to OpenMP controlled buffers

Future directions / Steps / Questions

- How to enable the MPI library to gain information about OpenMP's target memory allocations?
- How to enable the MPI library to map vendor agnostic and vendor specific allocations?



Topic: Language Bindings

Current State in MPI and OpenMP

- MPI currently supports C and Fortran up to F08
- C++ removed in MPI 3.0 as the interface
- OpenMP currently supports C, C++ and Fortran up to F08

Challenges

- New languages are in demand
- Renewed interest in C++
- Subset of API for lightweight execution?

Accelerator Bindings for MPI Partitioned APIs

CUDA, SYCL and ROCm Language Bindings Under Exploration

- Consequence of moving partitioned triggers to GPU
- Opens many cans of worms (What is a process? Which bindings? …)

```
int MPI_Psend_init(const void *buf, int partitions, MPI_Count count,
              MPI_Datatype datatype, int dest, int tag, MPI_Comm comm,
              MPI_Info info,MPI_Request *request)

int MPI_Precv_init(void *buf, int partitions, MPI_Count count,
              MPI_Datatype datatype, int source, int tag, MPI_Comm comm,
              MPI_Info info, MPI_Request *request)

int MPI_[start,wait][_all](...)
```

Keep host only

Add device bindings

```
__device__ int MPI_Pready(int partition, MPI_Request request)

__device__ int MPI_Pready_range(int partition_low, int partition_high, MPI_Request request)

__device__ int MPI_Pready_list(int length, const int array_of_partitions[], MPI_Request
request)

__device__ int MPI_Parrived(MPI_Request request, int partition, int *flag)
```

Separation of Bindings from MPI Concepts

New user communities use new/other languages

- C++, Python, Java, …
- MPI should be available to them, in their „native usage pattern“, not as C wrappers

One idea

- Split the MPI standard into semantics and bindings
- One central semantics document
- One document per language to describe the mapping

Side effects

- Would make us think more about the details of the semantics of the standard

- Question: what does this mean for scripted languages?

Early discussions / New working group

Topic: Language Bindings

Current State in MPI and OpenMP

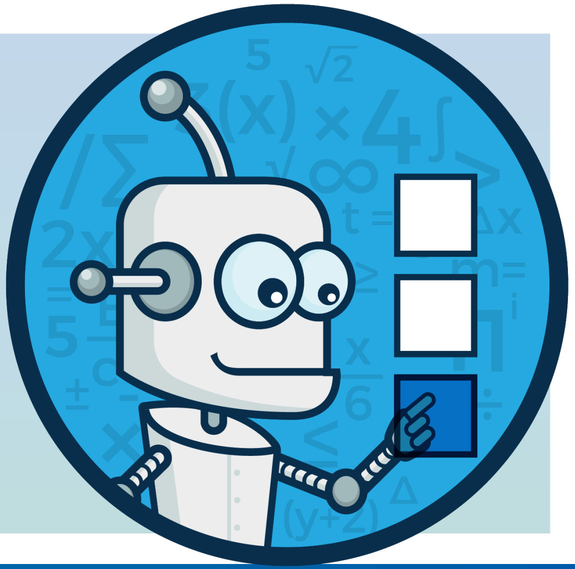
- MPI currently supports C and Fortran up to F08
- C++ removed in MPI 3.0 as the interface
- OpenMP currently supports C, C++ and Fortran up to FXX

Challenges

- New languages are in demand
- Renewed interest in C++
- Subset of API for lightweight execution?

Future directions / Steps / Questions

- Any dependencies/need for a C++ interface?
- Use of lightweight MPI calls inside target regions?
- Coordination on future languages between MPI and OpenMP needed?



Moving MPI and OpenMP forward in Harmony

State of the Art: Using both models together

- Basics are simple, as they are independent
- MPI needed thread support, which is give since MPI 2.x
- Used in many hybrid codes

A "next-to-each-other" seems long-term infeasible

- Need more information exchange
- Need access to each other's progress and state

Areas of concerns/improvements

- Shard thread semantics
- Support for asynchronous runtime behavior
- Lightweight communication from within a thread
- Memory allocation types
- Language bindings

Credit: DALL-E

Moving MPI and OpenMP forward in Harmony

State of the Art: Using both models together

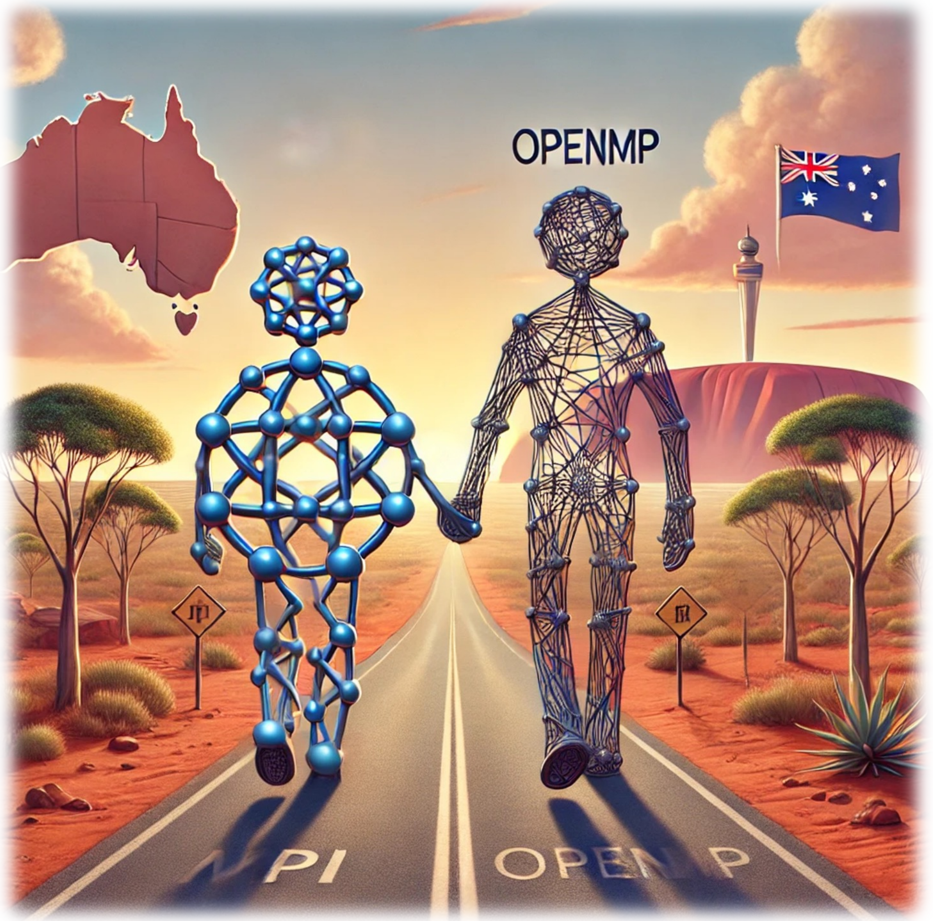
- Basics are simple, as they are independent
- MPI needed thread support, which is give since MPI 2.x
- Used in many hybrid codes

A ”next-to-each-other” seems long-term infeasible

- Need more information exchange
- Need access to each other’s progress and state

Areas of concerns/improvements

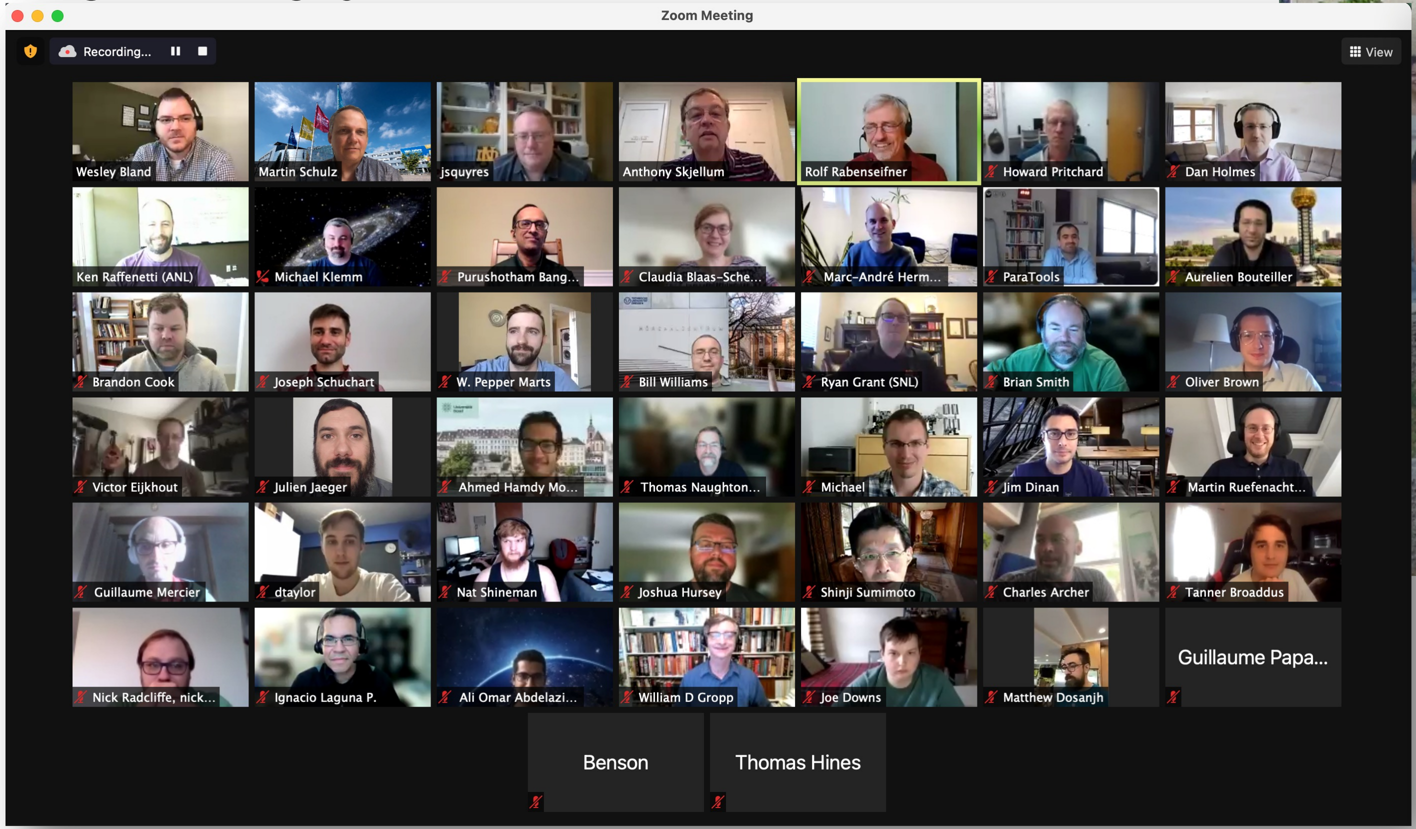
- Shard thread semantics
- Support for asynchronous runtime behavior
- Lightweight communication from within a thread
- Memory allocation types
- Language bindings



Credit: DALL-E

It takes a team, or rather teams …

The MPI Forum



The TUM CAPS Team

Bayerisches Staatsministerium für Wissenschaft und Kunst

Bayerische Forschungsstiftung

SPONSORED BY THE

Federal Ministry of Education and Research

Federal Ministry for Economic Affairs and Climate Action

Moving MPI and OpenMP forward in Harmony

State of the Art: Using both models together

- Basics are simple, as they are independent
- MPI needed thread support, which is give since MPI 2.x
- Used in many hybrid codes

A "next-to-each-other" seems long-term infeasible

- Need more information exchange
- Need access to each other's progress and state

Areas of concerns/improvements

- Shard thread semantics
- Support for asynchronous runtime behavior
- Lightweight communication from within a thread
- Memory allocation types
- Language bindings

Credit: DALL-E

https://www.ce.cit.tum.de/caps/
schulzm@in.tum.de