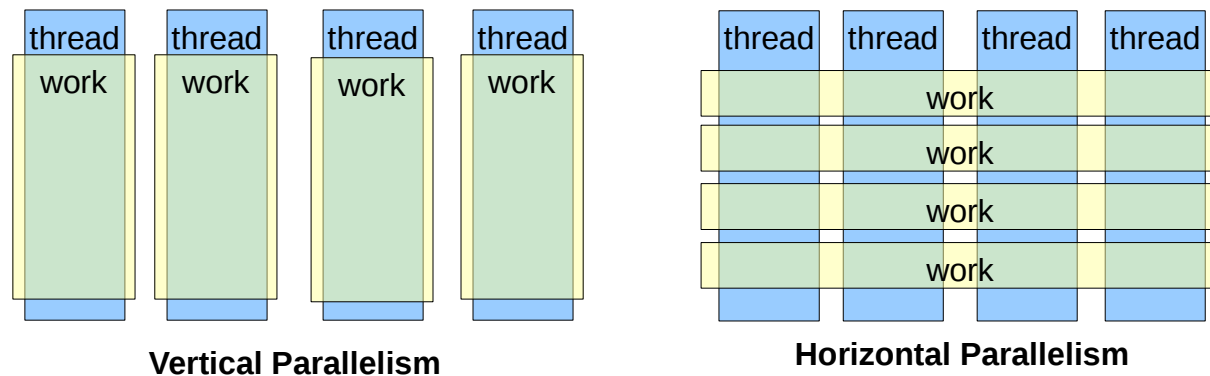Introductory Discussion of Extension to
"MPI3: Hybrid Programming"
for addition parallelism models

# Introduction



**Vertical Parallelism**

**Horizontal Parallelism**

- Horizontal vs. Vertical Parallelism:
  - Vertical Parallelism: One communication using one thread, many communications at the same time resulting in parallelism.
  - Horizontal Parallelism: One communication using many threads.
- Current "MPI3: Hybrid Programming" proposal supports/promotes vertical parallelism, but not horizontal parallelism.

# Introduction (cont.)
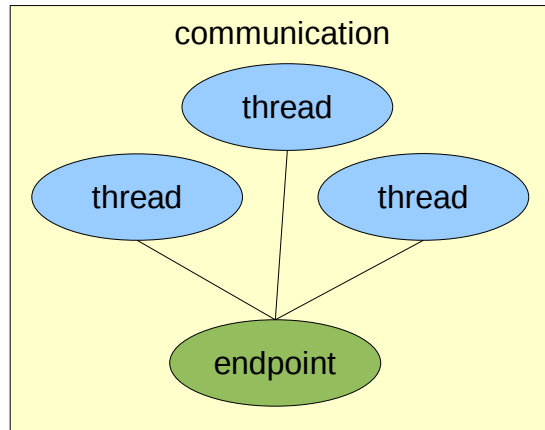


Multiple threads per endpoint
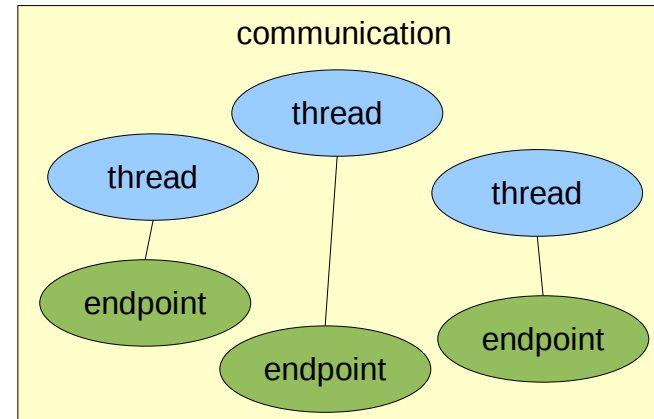


Multiple endpoints per communication

- Pictured above are two methods of applying multiple threads to a single communication.
  - Multiple threads per endpoint:
    - May require relaxing MPI_THREAD_MULTIPLE requirement for attaching multiple threads to an endpoint.
  - Multiple endpoints per communication:
    - More endpoints per communication, more potential parallelism.
- Mixing multiple endpoints with multiple threads per endpoint in the same program:
  - May require more threads, if each endpoint needs more than one thread to achieve maximum parallelism.
  - May require more changes in thread-endpoint relationships (detach/attach), with associated program logic, in order to move threads to where they are needed.

# Introduction (cont.)

- Both cases require some sort of grouping such that a communication call can tell whether it is acting alone (vertical) or participating in the same operation with other threads/endpoints (horizontal).
- It is also important for all participants to know what endpoints the remote nodes will use, in cases where communications must specify (choose) the remote endpoint(s) as targets.
- Assumption: Multiple endpoints per communication is the way to provide horizontal parallelism:
  - Endpoints are the "unit of parallelism" for communications.
  - Endpoints may operate independently.
  - For horizontal parallelism, the actual parallelism is either units of software (e.g. combining data from buffers) or hardware (e.g. injection and reception FIFOs on a hardware network interface), or a combination of both.
  - Using multiple threads on one endpoint (the alternative) to make communications calls together requires MPI_THREAD_MULTIPLE which horizontal parallelism does not necessarily require.
    - since all threads are cooperating on a single task rather than all trying to do a similar task in different contexts, the implementation may not need the same level of "thread safeness".

# Possible Solution #1: Explicit endpoint grouping

- MPI_Endpoint_join(group_id, num_members)
  - Adds endpoint to a group.
  - Any communications call on an endpoint in group may use all endpoints in the group.
  - Probably followed by a barrier to ensure the full set of endpoints is available before starting communications.
  - An endpoint can only join one group at a time.
    - No changes are required to MPI communications calls – the caller's endpoint directly "points" to the group of endpoints available for use.
  - A less-formal group membership might be possible.
- MPI_Endpoint_leave()
  - Blocking, collective call – does not return until all members call MPI_Endpoint_leave <u>and</u> all currently outstanding communications on endpoint has complete.
    - Since each thread (agent) does not know what other communications may have used it's endpoint, all endpoints must make progress until all members of the group have completed their communications.
- Any communications started/performed between the join and leave may use (at the implementation's discretion) any of the endpoints in the group.
- The endpoint used to start a communication must be the endpoint at which the completion occurs – e.g. the request generated must work in an MPI_Wait on the same endpoint.
- No knowledge of remote endspoints' participation, beyond communicator.

# Possible Solution #1: Explicit endpoint grouping (cont.)

- An implementation is free to ignore the endpoint group and never use more than the caller's endpoint.
  - The implementation must provide the expected behavior of join and leave, though.
- Outside of a join/leave region, the endpoint is isolated (no horizontal parallelism).
  - If a communications call is made without join/leave, that communication uses only the caller's endpoint.
    - A non-blocking communication ("A") followed by a join/communicate/leave block of code results in "A" being completed as part of the leave.
    - Could "A" use other threads/endpoints in the group then?
- Programs without delineated communication phases cannot use horizontal parallelism.
  - Without a defined group of threads/endpoints the communication cannot be certain that endspoints used will make progress.
  - Since join/leave forms a barrier, the program must be aware of compute/communicate phases.
  - Although, a program could have some threads perform a join and then continue with computation until calling leave at an appropriate time.
    - The computation would probably not be impaired, since the threads are not used by any work assigned to endpoints until calling the progress engine (in the leave).
    - However, the communications would probably not complete until all compute threads called leave.

# Possible Solution #1: Explicit endpoint grouping (cont.)

Discussion in MPI Forum Trac Ticket #208

Example:

```
#pragma omp parallel num_threads(num_endpoints)
{
    x = omp_get_thread_num();
    MPI_Endpoint_attach(endpoints[x]);
    /*
     * ... computation ...
     */
    MPI_Endpoint_join(group_id, num_endpoints);
    #pragma omp barrier
    /*
     * ... communication with horizontal parallelism ...
     */
    MPI_Endpoint_leave();
    /*
     * ... more computation? ...
     */


    /*
     * ... communication with (at-most) vertical parallelism ...
     */
    MPI_Endpoint_detach();
}
```

# Other Possible Solutions

2. Agents (or threads?) group together on the same communications call by explicit (new) parameters to all calls.
   - Discussion in MPI Forum Trac Ticket #209
   - Requires all MPI communication call signatures to change, for call-grouping parameters.
   - Probably not a practical solution: The level of change is probably prohibitive.
3. Multiple threads attach to the same endpoint, one thread makes communication call while the rest make some specialized "helper" call.
   - Discussion in MPI Forum Trac Ticket #210
   - Without multiple endpoints per communication, this may restrict how much parallelism an implementation may provide.
   - May require change of meaning to MPI_THREAD_MULTIPLE or change to meaning of multiple threads attached to one endpoint.
   - The additional threads are dedicated to communications for some period of time. Requires coordination (compute/communicate phases) to reclaim threads.
   - Could possibly assume that all attached threads will participate, but that breaks the ability to attach multiple threads to an endpoint for other purposes (what are those?)
   - Probably not a practical solution: Does not allow multiple endpoints to participate and may require MPI_THREAD_MULTIPLE change.

# Other Proposed Solutions (cont.)

4. Threads that are "available" (idle, etc) makes themselves available by attaching as "helper" threads. This attach does not return until a different thread calls the "detach helper" function.
   - Discussion in MPI Forum Trac Ticket #211
   - Threads performing communications (not attached as helpers) may wish to "barrier" until all (some, any, ...) helpers attach, but no mechanism exists.
   - There are complications with coordinating the detach.
   - There are complications with providing symmetry.
     - Threads cannot both perform communication and act as helper threads, at least not easily.
   - Probably not a practical solution: Solution #1 is more flexible.
5. All endpoints must make progress if any endpoint does communication.
   - Discussion in MPI Forum Trac Ticket #212
   - This forces coordination between threads about communication phases, not all application models use strictly-defined compute/communicate phases.
   - Or, requires additional threads to be dedicated to making progress on endpoints.
   - MPI does not currently allow "progress" without request objects, might require some specialized (generalized) request object(s).
   - Probably not a practical solution: coordination of progress at endpoints/threads that do not otherwise perform communication is problematic.