Endpoint Proposal

Ticket #288

Goal

- Support model with multiple “MPI Processes” per node, and shared memory across processes
- Motivations:
  - Hard to increase message rate per MPI process
    - Very hard to parallelize send-receive matching (combination of ordering requirement with process & tag matching with don’t-cares)
  - May want to localize arrival place in NUMA nodes
  - Provide better interoperability with PGAS
    - UPC gets better performance when UPC threads are mapped onto pthreads. Rather than OS processes. Natural to have one MPI process per UPC thread. Same for CAF images.

Two approaches

- "Unix V like" shared memory segments (separate proposal)
  – Each MPI process has private memory by default
  - Unix V model is not supported by current shared memory programming languages and tools
  - Is often constrained on the amount of memory that can be shared
  - Adds additional cleanup chores
- Multiple "MPI processes" in one address space – All memory is shared by default
  - Can use OpenMP, TBB, UPC, etc. on each node with multiple "MPI processes" per node
- We present here second approach

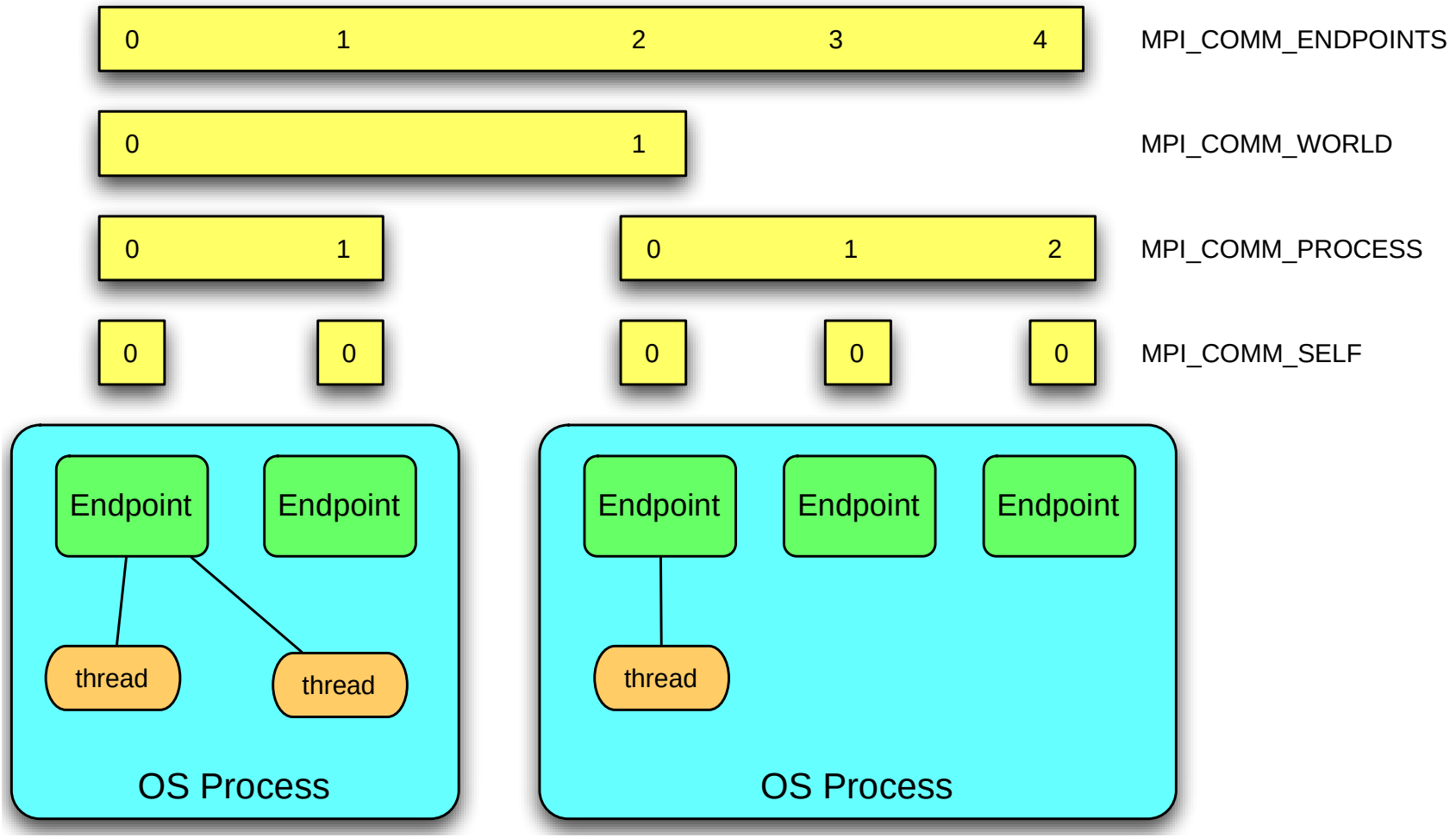
Terminology

- **OS process** (or Shared Address Space)
- **Thread**
- **MPI Endpoint**: set of resources that support MPI calls
  - Each <rank, comm> pair is associated with one endpoint; for each communicator, different ranks map to different endpoints; but <rank1,comm1> and <rank2, comm2> can be associated with same MPI endpoint.
- **MPI process**: an MPI endpoint with one or more threads attached to it
- New: *an OS process can contain multiple MPI processes*
  - (Not new, really)

Design Choices

- Number of endpoints at each OS process is determined before MPI starts running (e.g., by arguments to mpiexec)
  - Same as for number of MPI processes per node now
- Threads can dynamically attach to different endpoints within the same OS process
  - They migrate from one MPI process to another
- Newly created threads are attached to 1st endpoint in the OS process (the MPI_COMM_WORLD endpoint)
  - Compatible with current MPI
- *All MPI works unchanged, with "process" reinterpreted to mean "MPI process"*

Initial Communicators



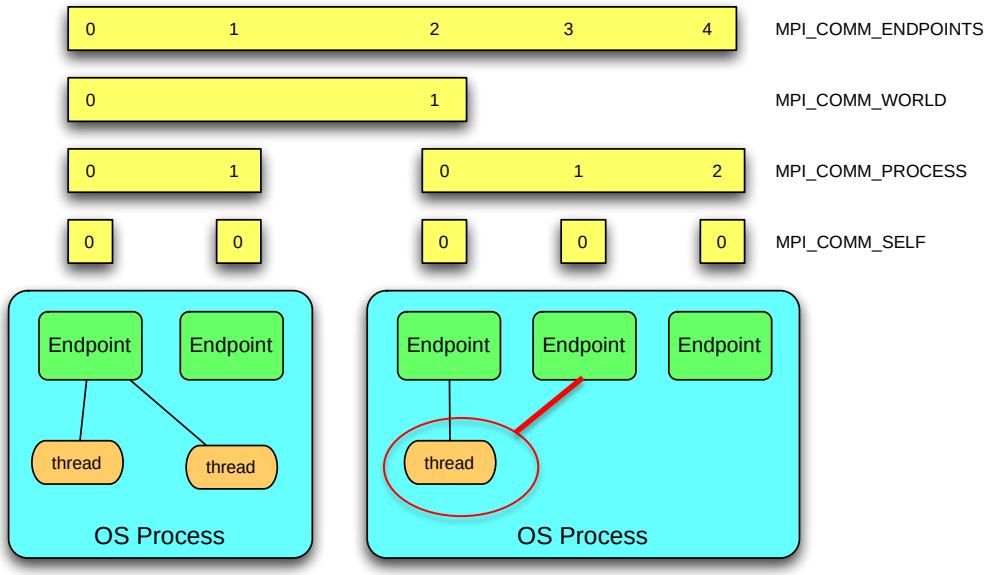
- *No change is needed in current MPI codes even if multiple endpoints are created at each OS process*

One New Function

- MPI_THREAD_ATTACH(rank, comm)
  - Thread detaches from current endpoint and attaches to new endpoint (migrates from one MPI process to another)
    - rank can be MPI_PROC_NULL – thread is not attached to any endpoint and cannot invoke MPI
    - Call is erroneous if new endpoint is not in local OS
  - A thread can be attached to only one endpoint (a thread belongs to at most one MPI process)
    - MPI calls by a thread refer to that endpoint (e.g., sender rank in MPI_SEND)
    - Many threads can be attached to same endpoint

Example

```
MPI_Thread_attach(1,
    MPI_COMM_PROCESS);
```




```
MPI_Send(…,
    MPI_COMM_PROCESS)
// sender rank = 1
```

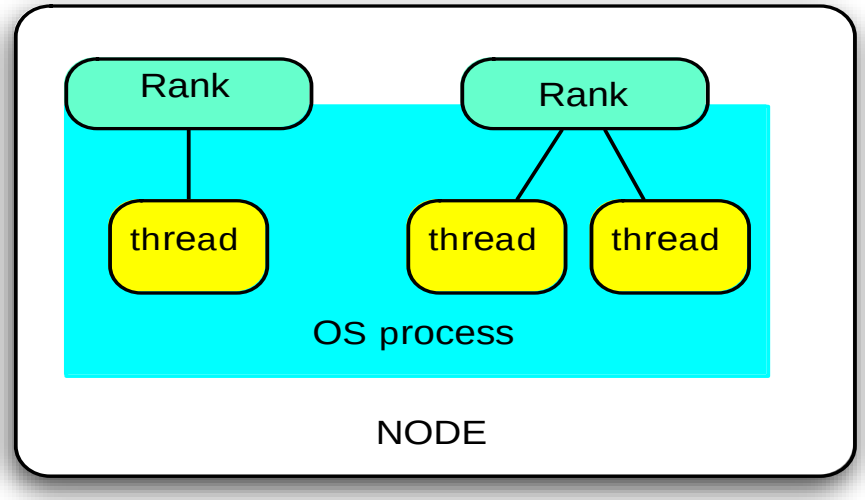
```
MPI_Send(…,
    MPI_COMM_ENDPOINT)
// sender rank = 3
```

Thread attaches to an endpoint (is in an MPI process); the endpoint can have different ranks in different communicators

FAQ

- *Any other changes?*
  - New info argument for MPI_COMM_SPAWN and MPI_COMM_SPAWN_MULTIPLE
  - INIT and FINALIZE invoked once per OS process, not once per endpoint
- *What happens if no thread is attached to an endpoint*
  - Same as when thread is attached but does not invoke MPI
- *Can an MPI nonblocking call be started on one endpoint and ended on another?*
  - NO. but as in current MPI, a nonblocking call started by one thread in an MPI process (i.e., on an endpoint) can be finished by another thread *in the same MPI process* (same endpoint)

Implementation



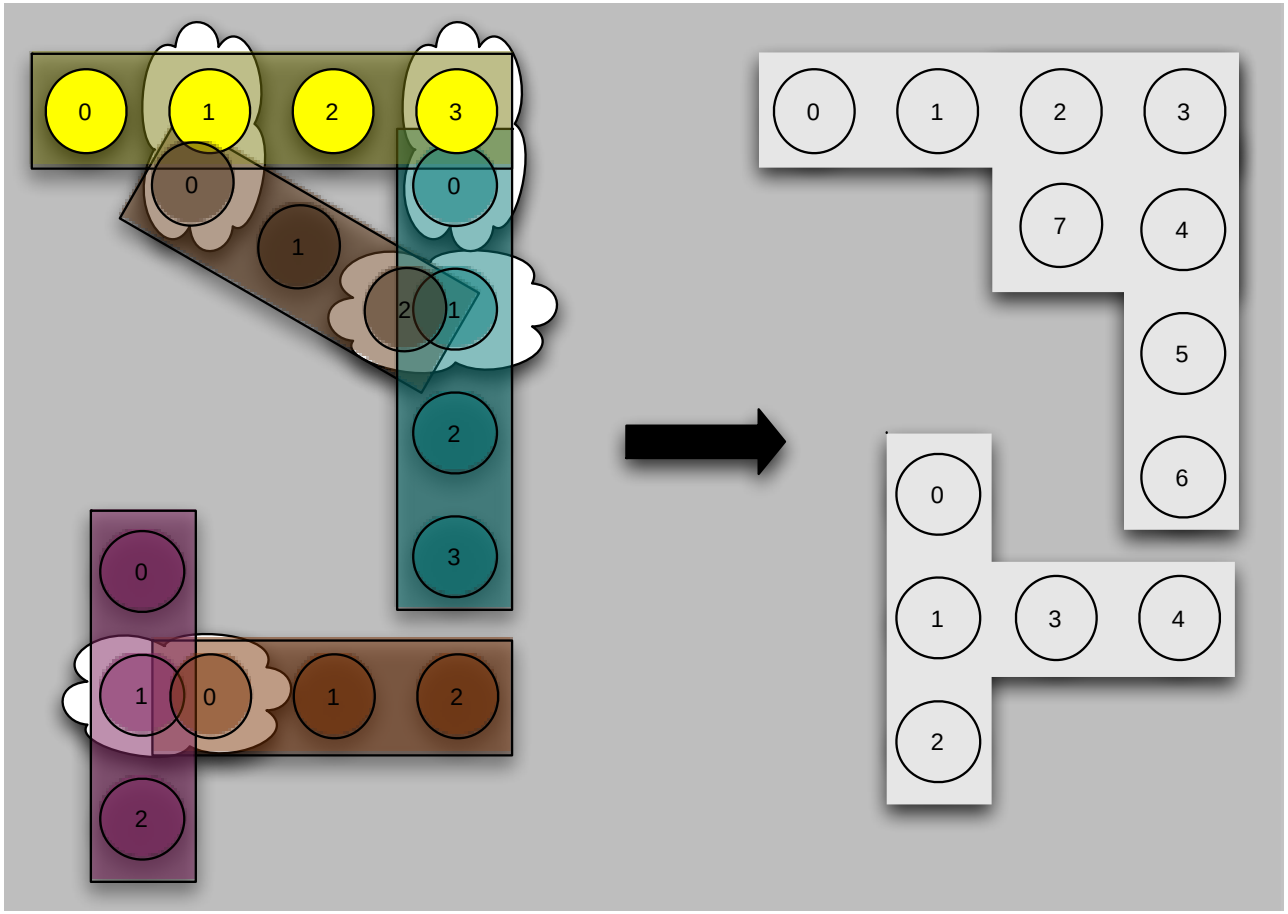
- One change only: keep for each thread a reference to the endpoint the thread is attached to
  - Thread-private memory or table indexed by thread_id

  (replace per process variable by per thread variable)

Communicator Merge Proposal

Ticket #289

Goal: Merge Communicators



Rationale

- Hard to do now:
  - Can merge 2 partially overlapping communicators using MPI_INTERCOMM_CREATE; MPI_INTERCOMM_MERGE
  - Need to break symmetry between two communicators (probabilistic algorithm)
- Useful: E.g., “No endpoint” MPI code calls endpoint-using library; invocation passes comm argument; invoked processes merge comm with MPI_COMM_PROCESS

New Function

- MPI_COMM_MERGE(comm1, comm2, newcomm)
  - Invoked by all MPI processes in the union of the communicators
  - MPI processes that belong to two communicators pass two comm arguments; the others pass one.
  - New communicator returned at all invoked MPI processes; one disjoint communicator for each connected component
  - New ranks are arbitrary