MPI_T Entities & Sets

Sidedocument discussion

Intention

- Define semantically consistent set of MPI_T entities
  - Tools should be able to rely on specified semantics and inter relations
- Have specific sets with defined names (in MPI Namespace)
  - Would MPI_(CVAR|PVAR|EVENT)_… be acceptable?
    - Should add namespace reservation into the standard and then define specific names in side-doc
- Specify sets as big as needed but as small as possible
  - Ensure that not entire set are never implemented due to problems with a single entity in the set

MPI Peruse via MPI_T

- Do we still want a full "MPI Peruse" equivalent?
- What is implementable/not in each of MPICH/OMPI/…?
- What is an implementable/modern version of the same?
- Is MPI_T the wrong tool for the job (e.g. lower level tools interfaces?)
- Tools wiki notes: https://github.com/mpiwg-tools/tools-issues/wiki/MPI_T-semantics-side-document(s)

Peruse Subset to track (multiple) transfers for a message?

- MPI_T_EVENT_(REQUEST_)TRANSFER_BEGIN
  - A request-related transfer begins
- MPI_T_EVENT_(REQUEST_)TRANSFER_END
  - A request-related transfer ends
- Which info needs to be part of the event to successfully track such messages
  - How to track point-to-point messages that a part of a collective communication algorithm?
  - EduMPI visualizes transfers part of non-blocking collectives using PVAR info (available in Open-MPI) (see next slide)
- How is blocking procedure handling different from non-blocking?
  - Are different event sets needed to track these?

Support for tracking non-blocking collectives (EduMPI)

- Currently done via PVARs defined for Open-MPI
  - Huge portability improvement if available elsewhere
- Lifetime problem with Requests
  - Binding done with an active handle
  - Requests may be too short lived
    - Race-condition when binding

GI paper

Euro-Par 25

Entity set to track handles (like requests, communicators)

- MPI_T_EVENT_COMM_CREATED
- MPI_T_EVENT_<handle>_FREEING

- PVAR registration possible in event handlers
- Could track "short-lived" handles (such as requests)
- Could track "non-standard" handles (such as specific communicators created via MPIX_ procedures (such as those available on BlueGene)

Maybe specify in main doc that mechanism to consistently bind to requests needs to be provided (entity set with Create/PVAR/Free set in sidedoc)

Is there still a problem with how binding is defined in the main doc?

Entity set to track states of requests

- MPI_T_EVENT_REQUEST_ACTIVATED
- MPI_T_EVENT_REQUEST_COMPLETED
- MPI_T_EVENT_REQUEST_CANCELLED
- MPI_T_EVENT_REQUEST_FREED

- Enables tools to track all states

General

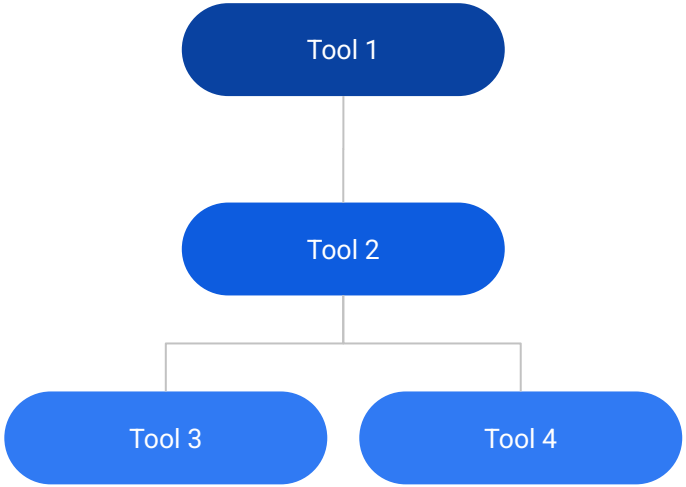
- Tracking request lifetimes/binding to (non-persistent) requests is stupid and often wrong
  - Add request create/destroy events to allow binding requests to MPI_T objects
  - Still need to solve the dummy request case
  - Add create/destroy events for other binding targets for symmetry?
    - What are the performance considerations?
    - What are the tradeoffs between “can do everything a tool needs via MPI_T” and “don’t duplicate ways to track things”?
    - What are the “holes” in MPI_T/should Cvars be usable to work around problems?

QMPI Recap

Open Questions

- Static/dynamic tools list decision
  - If dynamic, there should be a prototype somewhere *inside* an implementation w/benchmarks to compare against MPICH+static
  - Start with static w/o foreclosing dynamic
- skipping to impl: mechanism, semantics?
- Sandboxing via sessions
  - Should a QMPI tool be able to explicitly filter by session?
    i. Some way to ID a session at the tools level (MPI_Foo_GetSession?)
    ii. Add session attributes and pass them down?
    iii. Add session attributes and add **QMPI** accessors for session from handles to avoid the law of demeter problems/propagation to lower tools?
  - Should a QMPI tool stack be able to be different across sessions?
    i. Separate tools callback at creation to allow tools to register?
    ii. QMPI function to allow registration?
- What is the proper granularity of a tools stack's attachment?
  - How to specify such stacks?
  - How do we specify "something different from everything that's registered" as a tools stack?
- Formalize the concept of a “bubble”?

Static vs. dynamic tools



Tool 2 decides which of tools 3 and 4 to call at runtime

Static: tools (2,3,4) are to QMPI one unitary tool, and tools 3 and 4 register with tool 2 (possibly via a QMPI-like mechanism but out of our scope)

Dynamic: tools (2,3,4) are all QMPI tools and tool 2 decides its successor function at runtime *somehow* (again OOS but it needs to be able to query the pointers for tools 3 and 4)

Bindings

Session/Group/Comm 1

Session/Group/Comm 2

- Do we filter in each tool or let the implementation do some work?
- Where do we want to bind tools (as tool users/developers)?
- Where must we bind tools (from implementor POV)?
- Do we specify (1,2,3,4) and let 2/4 ignore L/R context?
- Do we specify (1,2,3) and (1,3,4) as valid tool stacks and let (WLOG) sessions pick which one to use (mechanism? Info key?)