Generalized Requests

The current definition

- They are defined in MPI 2 under the hood of the chapter 8 (External Interfaces)
- Page 166 line 16

> The objective of this MPI-2 addition is to allow users of MPI to be able to create new nonblocking operations with an interface similar to what is present in MPI. This can be used to layer new functionality on top of MPI.

More info

Such an outstanding nonblocking operation is represented by a (generalized) request. A fundamental property of nonblocking operations is that progress toward the completion of this operation occurs asynchronously, i.e., concurrently with normal program execution. Typically, this requires execution of code concurrently with the execution of the user code, e.g., in a separate thread or in a signal handler. Operating systems provide a variety of mechanisms in support of concurrent execution. MPI does not attempt to standardize or replace these mechanisms: it is assumed programmers who wish to define new asynchronous operations will use the mechanisms provided by the underlying operating system.

MPI Role

- The generalized requests are entirely managed by the application/library except for one state. The completion is managed by MPI, the application should use MPI_GREQUEST_COMPLETE to advertise the completion of the generalized request.

Generalized requests

MPI_GREQUEST_START(query_fn, free_fn, cancel_fn, extra_state, request)

- IN **query_fn** callback function invoked when request status is queried (callback function)
- IN **free_fn** callback function invoked when request is freed (callback function)
- IN **cancel_fn** callback function invoked when request is cancelled (callback function)
- IN **extra_state** extra state
- OUT **request** generalized request (handle)

The query function

- typedef int MPI_Grequest_query_function(
  void *extra_state, MPI Status *status);
- **query_fn** function computes the status that should be returned for the generalized request. The status also includes information about successful/unsuccessful cancellation of the request. query fn callback is invoked by the MPI_{WAIT|TEST}_{ANY|SOME|ALL} call that completed the generalized request associated with this callback. Note that **query_fn** is invoked only after MPI_GREQUEST_COMPLETE is called on the request; it may be invoked several times for the same generalized request, e.g., if the user calls MPI_REQUEST_GET_STATUS several times for this request.

The free function

- typedef int MPI_Grequest_free_function(void *extra_state);
- **free_fn** function is invoked to clean up user-allocated resources when the generalized request is freed. **free_fn** callback is invoked by the MPI_{WAIT|TEST}_{ANY|SOME|ALL} call that completed the generalized request associated with this callback. **free_fn** is invoked after the call to **query_fn** for the same request.

The cancel function

- typedef int MPI_Grequest_cancel_function(
  void *extra state, int complete);
- **cancel_fn** function is invoked to start the cancellation of a generalized request. It is called by MPI_REQUEST_CANCEL(request). MPI passes to the callback function complete=true if MPI_GREQUEST_COMPLETE was already called on the request, and complete=false otherwise.

Possible improvement(s) …

- 15 January 2008 2:30PM - 3:30 PM in parallel with the non-blocking collective session.
- Euro PVM/MPI 2007 :“Extending the MPI-2 Generalized Request Interface” Robert Latham, William Gropp, Robert Ross, and Rajeev Thakur

Possible improvements - I

- They propose an MPIX_GREQUEST_START function similar to MPI_GREQUEST_START, but taking an additional function pointer (**poll_fn**) that allows the MPI implementation to make progress on pending generalized requests.
- “When the MPI implementation tests or waits for completion of a generalized request, the poll function will provide a hook for the appropriate AIO completion function. All test and wait routines (MPI_{TEST|WAIT}_{ALL|ANY|SOME}) begin by calling the request’s user-provided **poll_fn**. For the wait routines, the program continues to call **poll_fn** until either at least one request completes (wait, waitany, waitsome) or all requests complete (wait, waitall).”

Possible improvements - II

- “We can give implementations more room for optimization if we introduce the concept of a generalized request class. MPIX_REQUEST_CLASS_CREATE would take the generalized request **query**, **free**, and **cancel** function pointers already defined in MPI-2 along with our proposed **poll** function pointer. The routine would also take a new “**wait**” function pointer. Initiating a generalized request then reduces to instantiating a request of the specified class via a call to MPIX_GREQUEST_CLASS_ALLOCATE.”

The short term plan

- Continue on the Proposal II: add a “class”, find a better name, add a poll and a wait function, insure similarities between their arguments, send a proposal on the mailing list (in about a week).
- Open question: How the progress is now made and when ?

Progress modes

- 3 acceptable progress modes:
  - Signal like approach: the library use MPI_GREQUEST_COMPLETE to mark the generalized request as completed
  - Non-blocking approach: poll can be used to check for any progress (based on an array of generalized requests)
  - Blocking approach: the library is allowed to block until one (some) of the generalized requests passed in the array argument is completed