Nonblocking
Process Creation & Management

Josh Hursey

Open Systems Lab.

jjhursey@osl.iu.edu

INDIANA UNIVERSITY

PERVASIVE TECHNOLOGY INSTITUTE

Adding MPI_Request to all functions

For all of the nonblocking routines described in this section an MPI_REQUEST object is returned. All of the completion calls (e.g., MPI_WAIT, MPI_TEST) are supported for these nonblocking routines. Upon returning from a completion call, the MPI_ERROR field of the MPI_STATUS object is set appropriately. The values of the MPI_SOURCE and MPI_TAG fields are undefined. The parameters marked as OUT should not be accessed until the request has been completed by one of the completion calls.

Example

```
int foo(int num_children) {
  for(i = 0; i < num_children; ++i) {
    MPI_Icomm_spawn("myapp", argv, 1, info, 0,
                    MPI_COMM_SELF,
                    &intercomm[i], &array_of_statuses[i],
                    &(requests[i]));
  }

  prepare_work_units();

  MPI_Waitall(num_children, requests, statuses);

  begin_work();
}
```

Matching

Matching blocking and nonblocking MPI_ICOMM_SPAWN,
MPI_ICOMM_SPAWN_MULTIPLE,
MPI_ICOMM_ACCEPT,
MPI_ICOMM_CONNECT, and
MPI_ICOMM_JOIN
operations is ***not*** allowed.

> **Rationale**: The algorithms implemented for the blocking and nonblocking versions of these operations may not be equivalent for reasons of efficiency.

```
if(0 == rank) { MPI_COMM_ACCEPT(...); } /*** BAD ***/
else          { MPI_ICOMM_ACCEPT(...); MPI_WAIT_ALL(); }
```

MPI_ICOMM_SPAWN{_MULTIPLE}

MPI_ICOMM_SPAWN(command, argv, maxprocs, info, root, comm, intercomm, array_of_errcodes, ***request***)

MPI_ICOMM_SPAWN_MULTIPLE(count, array_of_commands, array_of_argv, array_of_maxprocs, array_of_info, root, comm, intercomm, array_of_errcodes, ***request***)

IN command	name of program to be spawned
IN argv	arguments to command
IN maxprocs	maximum number of processes to start
IN info	key-value pairs (e.g., where and how to start the processes)
IN root	rank of process in which previous arguments are examined
IN comm	intracommunicator containing group of spawning processes
OUT intercomm	intercommunicator between original group and the newly spawned group
OUT array_of_errcodes	one code per process
OUT request	***process creation request***

MPI_ICOMM_CONNECT/ACCEPT

MPI_ICOMM_CONNECT(port_name, info, root, comm, newcomm, ***request***)
IN port network address
IN info implementation-specific information
IN root rank in comm of the root node
IN comm intracommunicator over which call is collective
OUT newcomm intercommunicator with client as remote group
OUT request connection establishment request

MPI_ICOMM_ACCEPT(port_name, info, root, comm, newcomm, ***request***)
IN port port name
IN info implementation-specific information
IN root rank in commm of the root node
IN comm intracommunicator over which call is collective
OUT newcomm intercommunicator with client as remote group
OUT request connection establishment request

Connection Management

MPI_IOPEN_PORT(info, port_name, ***request***)

MPI_ICLOSE_PORT(port_name, ***request***)

MPI_IPUBLISH_NAME(service_name, info, port_name, ***request***)

MPI_IUNPUBLISH_NAME(service_name, info, port_name, ***request***)

MPI_ILOOKUP_NAME(service_name, info, port_name, ***request***)

MPI_ICOMM_DISCONNECT(comm, ***request***)

MPI_ICOMM_JOIN(fd, intercomm, ***request***)

/ There is not a nonblocking **MPI_COMM_GET_PARENT** */*

MPI_CANCEL(&request)

- MPI 2.2: Chapter 3.8: Probe and Cancel

> “... marks for cancellation a pending, nonblocking communication operation (send or receive). The cancel call is ***local***. It ***returns immediately***, possibly before the communication is actually canceled. It is still necessary to complete a communication that has been marked for cancellation...”
>
> “Either the cancellation succeeds, or the communication succeeds, but ***not both***.”

- MPI_TEST_CANCELLED(status, flag)

Canceling NB Communicator Creation

MPI_ICOMM_ACCEPT(..., request)

If a MPI_REQUEST for MPI_ICOMM_ACCEPT is marked for cancellation using MPI_CANCEL, then it must be the case that either

- the operation completed normally, in which case the connection is established, or
- that the operation is canceled, in which case the any pending requests will be canceled returning MPI_ERR_NOT_CONNECTED to any processes waiting in MPI_COMM_CONNECT or MPI_ICOMM_CONNECT.

MPI_CANCEL can be called from ***any*** participating process.

Canceling NB Communicator Creation

MPI_ICOMM_CONNECT(..., request)

If a MPI_REQUEST for MPI_ICOMM_CONNECT is marked for cancellation using MPI_CANCEL, then it must be the case that either

- the operation completed normally, in which case the connection is established, or
- that the operation is canceled, in which case the connection is not established.

If the connection was establishing when the cancellation occurred then the matching MPI_COMM_ACCEPT or MPI_ICOMM_ACCEPT call may return MPI_ERR_NOT_CONNECTED.

MPI_CANCEL can be called from ***any*** participating process.

Canceling NB Communicator Creation

MPI_ICOMM_SPAWN(..., request)

If a MPI_REQUEST for MPI_ICOMM_SPAWN or MPI_ICOMM_SPAWN_MULTIPLE is marked for cancellation using MPI_CANCEL, then it must be the case that either the operation completed normally, in which case the processes launched and communicator was created, or that the operation is canceled, in which case the processes are not launched and the communicator is not created.

MPI_CANCEL can be called from ***any*** participating process.

Advice to implementors: The root can decide if it is able to or safe to cancel the request when it is notified of the cancelation request. If the processes have already started to be launched, the implementation is allowed to refuse the cancellation request if it is unable or unwilling to terminate the new processes. Alternatively, the implementation may decide that it is willing and able to terminate a newly launched process. Care should be taken when doing so since the process may cause side effects in the system. For example, if the launched process interacts with the file system before calling MPI_INIT this may influence already running processes.

Timeout MPI_Info Keyword (For MPI_COMM_ACCEPT/CONNECT)

The MPI 2.0 standard notes for MPI_Comm_connect in Chapter 5, Section 5.4.3 that:

> If the port exists, but does not have a pending MPI_COMM_ACCEPT, the connection attempt will eventually time out after an implementation-defined time, or succeed when the server calls MPI_COMM_ACCEPT. In the case of a time out, MPI_COMM_CONNECT raises an error of class MPI_ERR_PORT.
>
> Advice to implementors. The time out period may be arbitrarily short or long. However, a high quality implementation will try to queue connection attempts so that a server can handle simultaneous requests from several clients. A high quality implementation may also provide a mechanism, through the info arguments to MPI_OPEN_PORT, MPI_COMM_ACCEPT and/or MPI_COMM_CONNECT, for the user to control timeout and queuing behavior. (End of advice to implementors.)

Reserved Key Values: `timeout`

... The value is specified as a positive integer representing the timeout in units of MPI_WTICK. The timeout is defined as the time it takes for the entire operation to complete. If the value is set to 0 then the timeout is set to an MPI implementation defined limit. The keyword is only meaningful at the root.

> ***Rationale**: The server might stall for a variety of reasons (e.g., fault recovery, overloaded). The client may want to limit their sensitivity to slow servers. The timeout key allows them to cancel a blocking connection establishment operation.*

`MPI_ERR_NOT_AVAILABLE`	Server state is unknown
`MPI_ERR_NOT_RESPONDING`	Server is alive, but not responding
`MPI_ERR_NOT_CONNECTED`	Connection to server started, but could not be finished

Reserved Key Values: `timeout`

If MPI_ERRORS_RETURN is not set on the communicator then the MPI_ERR_NOT_RESPONDING, MPI_COMM_NOT_AVAILABLE, and MPI_ERR_NOT_CONNECTED are fatal as defined by the default of MPI_ERRORS_ARE_FATAL, even though the error code may have been caused by the timeout on the operation.

> ***Rationale**: It was recognized that the application, by setting the timeout key value, may expect that the errors caused by the timeout of the operation should not be fatal. However, by making this class of errors nonfatal, will cause previous standard conferment applications to break since functions that they assumed would only return MPI_SUCCESS would then return a wider class of errors.*

http://meetings.mpi-forum.org/

https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/FaultToleranceWikiPage