Proposed extension to "MPI3: Hybrid Programming" for support of horizontal parallelism
V0.2 Mar 9, 2010
Douglas Miller, dougmill@us.ibm.com

MPI_ENDPOINT_TEAM_CREATE(team_id, team_size, team_handle)

IN team_id	locally unique identifier of team to join
IN team_size	total number of members in team
OUT team_handle	handle describing team

The function creates a team to be used with subsequent JOIN calls. The call is made by all members of the team, but is non-blocking. *team_id* is an integer that is unique among all currently created teams, but may be re-used as long as the previous team using that ID was destroyed. [if the function is blocking and collective over members of the team, the need for *team_id* could be eliminated]

MPI_ENDPOINT_TEAM_DESTROY(team_handle)

IN,OUT team_handle	handle describing team

The function destroys the team. This function is called by all entities that created the team. It is non-blocking.

MPI_ENDPOINT_JOIN(team_handle)

IN team_handle	handle describing team

The function registers the calling agent (calling thread's endpoint) as an active participant in the team. The caller's endpoint may now be used by communications started by other members of the team. This call is typically followed by a barrier over the members of team, in order to ensure communications will have access to the full set of endpoints. It is an error to call this function from a thread that is not attached to an endpoint. This function may be called from more than one thread attached to the same endpoint, but an implementation may not use all threads. An endpoint may only be active in one team at a time.

MPI_ENDPOINT_LEAVE()

The function completes all outstanding communications for the team, then dissolves the team association with the endpoints. It is blocking and collective over the members of the team. Until all members reach (call) the LEAVE, any members in LEAVE are effectively in MPI_WAIT - i.e. they are calling the progress engine.

Usage and Examples

These functions denote a boundary for a region of horizontal parallelism. All communications performed within this region (by the members) may use all member endpoints (and threads) to perform the communication(s). The endpoint on which a communications is started (the actual communications call occurs) is the primary endpoint for that operation. It is expected that the destination, or remote members of the communicator, are also the primary endpoints for their respective remote nodes.

Any communication performed outside (without) JOIN/LEAVE will utilize only the calling endpoint. It is an error if all participants are not similarly involved, i.e. all must be in a JOIN/LEAVE or none are in JOIN/LEAVE.

[suppose a communication that will use multiple endpoints for the actual remote transfers - e.g. message striping or injection vs. reception FIFOs - in such a case we need to define how the remote endpoints are properly identified]

[what are the deadlock potentials?]

Example 1: OpenMP program with distinct compute/communicate phases

A simple example where one thread performs an allreduce but the rest of the threads lend themselves as helpers. The omp barrier is to ensure that all endpoints are available when the allreduce begins.

```
#pragma omp parallel num_threads(N) {
      t = omp_get_thread_num();
      MPI_Endpoint_attach(endpoints[t]);
      MPI_Endpoint_team team;
      MPI_Endpoint_team_create(0, omp_get_num_threads(), &team);
      /*
       * some computation may occur here...
       */
      MPI_Endpoint_join(team);
      #pragma omp barrier
      if (t == 0) {
            MPI_Allreduce(...);
      }
      MPI_Endpoint_leave();
      /*
       * more computation and/or communication
       */
      MPI_Endpoint_team_destroy(&team);
}
```

Example 2: Pthreads program with distinct compute/communicate phases

```
main(...) {
      ...
      /* N is the total number of threads to participate */
      for (x = 0; x < N - 1; ++x) {
            pthread_create(..., compute_only, ...);
      }
      compute_comm(N);
      ...
}

compute_only(...) {
      MPI_Endpoint_team team;
      MPI_Endpoint_team_create(0, N, &team);
      while (!done) {
            /*
             * perform computation phase here...
             */
            MPI_Endpoint_join(team);
            pthread_barrier_wait(...);
            MPI_Endpoint_leave();
      }
      MPI_Endpoint_team_destroy(&team);
}
```

```
compute_comm(...) {
      MPI_Endpoint_team team;
      MPI_Endpoint_team_create(0, N, &team);
      while (!done) {
            /*
             * perform computation phase here...
             */
            MPI_Endpoint_join(team);
            pthread_barrier_wait(...);
            MPI_Allreduce(...);
            MPI_Endpoint_leave();
      }
      MPI_Endpoint_team_destroy(&team);
}
```