HSplit: functions, implementation issues and usage

Inria TADaaM Team

1 Communicator/group creation functions

1.1 MPIX_COMM_SPLIT_TYPE

MPI_COMM_SPLIT_TYPE is part of MPI 3.1 and is a communicator management routine defined in Chapter 6 of the standard (Groups, Contexts, Communicators and Caching), section 6.4 (Communicator Management).

MPI_COMM_SPLIT_TYPE(comm, split_type,key,info,newcomm)

IN comm communicator (handle)

IN split_type type of processes to be grouped together (integer)

IN key control of rank assignment (integer)

IN info info argument (handle)

OUT newcomm new communicator (handle)

C Prototype :

```
int  MPI_Comm_split_type(MPI_Comm comm, int split_type, int key,
                         MPI_Info info, MPI_Comm *newcomm)
```

MPI 3.1 defines a single value for split_type : MPI_COMM_TYPE_SHARED, that allows the function to split the original communicator into subcommunicators, which encompass MPI processes that can create a shared memory region with the other member processes.

Current extension/implementation :

- Introduces/relies on a new split_type value : MPI_COMM_TYPE_HW_DOMAIN Behaviour description : the input communicator is split into subcommunicators, each of which encompasses MPI processes that do share resources in the underlying physical topology (e.g. a network switch, a physical node, a L3 cache, a L2 cache, a core, etc.). This sharing of resources is detected or known by the *implementation* or another subsystem.
- Uses several *keyvals* :

- MPI_HW_DOMAIN_TYPE : If this key is defined (in the info parameter passed to the function), MPI_COMM_SPLIT_TYPE can then perform a split operation on a *specific hardware level*, whose name is the value of the MPI_HW_DOMAIN_TYPE key. The levels names are not specified and are therefore implementation-dependent. Such names can be queried by the MPIX_GET_HW_DOMAIN_INFO and MPIX_GET_HW_TOPOLOGY_INFO routines. For instance, an HWLOC-based implementation could use the HWLOC types names as domain types :
  - HWLOC_OBJ_MACHINE
  - HWLOC_OBJ_PACKAGE
  - HWLOC_OBJ_CORE
  - HWLOC_OBJ_PU for hardware threads
  - HWLOC_OBJ_NUMANODE
  - HWLOC_OBJ_L1CACHE, ..., HWLOC_OBJ_L5CACHE
  - HWLOC_OBJ_GROUP for other hierarchy levels

  Simpler (shorter) names (e.g. Core, Socket, Machine) can be also used. In this way, names are still not specified in the MPI standard but can be used nevertheless. Despite that the names are not standard, the *way* of accessing a hardware level (that is, the sequence of MPI calls to achieve this) is standard and the same, regardless of the implementation and the names they internally define and use.
- MPI_HW_DOMAIN_NUM : This key is used internally and is never made available to the user. It indicates the number of subcommunicators produced after a split operation at a specific level of the hardware hierarchy. This information can then be used to distribute data among the newly created subcommunicators. The value can be obtained by the user with a call to MPIX_GET_HW_DOMAIN_INFO.
- MPI_HW_DOMAIN_RANK : This key is used internally and is never made available to tue user. It indicates the "rank" of a subcommunicator produced after a split operation at a specific level of the hardware hierarchy. This information can then be used to distribute data among the newly created subcommunicators. The value can be obtained by the user with a call to MPIX_GET_HW_DOMAIN_INFO.

Implementation guidelines An implementation can support (implement) this addition to MPI_COMM_SPLIT_TYPE in multiple ways which are all acceptable with regard to the defined behaviour (i.e. new communicators materialize the sharing of physical resources between processes) :

- If no valid communicator can be created, the value MPI_COMM_NULL is returned, and this value can be used to assess if the last level of the hierarchy has been reached. In particular, it is possible for an implementation to not produce subcommunicators by directly returning the value.
- An implementation can simply return MPI_COMM_NULL if it cannot/does not want to support this feature.
- An implementation can choose to create a new communicator for *every* level present in the underlying physical hierarchy.
- Alternatively, in order to avoid the creation of redundant objects, the implementation can decide that the group of MPI processes supporting a new subcommunicator can be a *strict* subset of the group supporting the input (parent) communicator. More specifically, a call to MPI_COMM_COMPARE(comm,newcomm) should return MPI_UNEQUAL in this case. In case of several possible levels, the implementation is free to return the level considered as the most

relevant, for instance the *highest* or the *lowest* level in the hierarchy. However, to maintain a certain degree of consistency with other functions described in this document (especially MPI_COMM_GET_MIN_HLEVEL), the *lowest* level is an appropriate candidate.

Usage There are two ways to use this function, the *undirected* mode and the *directed* mode.

— **In the *undirected* mode**, it is possible to capture the hierarchical nature of the underlying hardware by calling "recursively" MPI_COMM_SPLIT_TYPE with MPI_COMM_TYPE_HW_DOMAIN as the split_type value on newly created subcommunicators and *no info* is provided to guide the split operation. Code example :

```
  MPI_Comm newcomm[NLEVELS];
  MPI_Comm oldcomm = MPI_COMM_WORLD;
  int rank, idx = 0;
  while((oldcomm != MPI_COMM_NULL) && (idx < NLEVELS))
  {
    MPI_Comm_rank(oldcomm,&rank);
    MPI_Comm_split_type(oldcomm,
                        MPI_COMM_TYPE_HW_DOMAIN,
                        rank,
                        MPI_INFO_NULL,
                        &newcomm[idx]);
    oldcomm = newcomm[idx++];
  }
```

— **In the *directed* mode**, an info keyval has to be provided (along with the MPI_COMM_TYPE_HW_DOMAIN for the split_type value) in order to create a communicator corresponding to a *specific* hardware level in the hierarchy. The key name is MPI_HW_DOMAIN_TYPE and its value should be a level name. In order to obtain information about the physical hierarchy, that is, the number of levels exposed by the implementation (levels being redundant or not) as well as the various level names (values for standard-defined keys) the query function MPIX_GET_HW_TOPOLOGY_INFO must be called. This function uses keyvals whose names are MPI_HW_LEVEL0,…,MPI_HW_LEVELN. Code example :

```
{
 MPI_Comm out_comm;
 MPI_Comm oldcomm = MPI_COMM_WORLD;
 MPI_Info info;
 int rank, idx, flag;
 char *resource_type = NULL;
 char str[MPI_MAX_INFO_VAL-1];
 char str2[MPI_MAX_INFO_VAL-1];

 MPI_Comm_rank(oldcomm,&rank);
 MPI_Info_create(&info);

 MPIX_Get_hw_topology_info(&numlevels,info);
 fprintf(stdout,"Number of levels available: %i\n",numlevels);
```

```
  /* could be retrieved with MPI_Info_get_nthkey instead */
  for(idx = 0 ; idx < numlevels; idx++){
    sprintf(str,"MPI_HW_LEVEL%d",idx);
    MPI_Info_get(info,str,MPI_MAX_INFO_VAL-1,str2,&flag);
    if (flag)
      fprintf(stdout,"%s type is %s\n",str,str2);
  }

  /* Let us suppose that Socket is an available level */
  MPI_Info_set(info,"MPI_HW_DOMAIN_TYPE","Socket");
  MPIX_Comm_split_type(oldcomm,MPI_COMM_TYPE_HW_DOMAIN,rank,info,&out_comm);

  /* Or split at level 2 in the hierarchy */
  MPI_Info_get(info,"MPI_HW_LEVEL2",MPI_MAX_INFO_VAL-1,str2,&flag);
  if(flag){
    MPI_Info_set(info,"MPI_HW_DOMAIN_TYPE",str2);
    MPIX_Comm_split_type(oldcomm,MPI_COMM_TYPE_HW_DOMAIN,rank,info,&out_comm);
  }
}
```

Relationship with mapping/binding of processes The mapping and binding of MPI processes onto physical resources must be taken into account for subcommunicators creation. Indeed, the *deepest* hardware level in the hierarchy corresponding to a subcommunicator should always correspond to resource the calling MPI process is bound to. For instance, if a process is bound to a certain cache level, no information below this cache level can be returned, as the MPI process can possibly use any of the caches below the level it is bound to. Any attempt to create a subcommunicator corresponding to a hardware level below the level the MPI process is bound to must return MPI_COMM_NULL.

1.2 MPIX_GET_HW_DOMAIN_NEIGHBOURS

MPIX_GET_HW_DOMAIN_NEIGHBOURS(comm,hops,metric,newgroup)

IN comm communicator (handle)

IN hops number of hops in the topology, neighborhood extent (integer)

IN metric neighborhood type (integer)

OUT newgroup new group (handle)

C Prototype :

```
int MPIX_Get_hw_domain_neighbours(MPI_Comm comm, int hops,
                                  int metric,MPI_Group *newgroup)
```

Current extension/implementation :

— Creates a *group* instead of a *communicator* since it is probably a local operation and the information should not propagate to other processes in the input communicator.

— Can possibly create neighborhood of various types, for instance *memory/computing* neighborhoods, *network* neighborhoods, etc.

2 Query functions

2.1 MPIX_GET_HW_TOPOLOGY_INFO

`MPIX_GET_HW_TOPOLOGY_INFO(numlevels,info)`

OUT `numlevels` number of levels in the hardware hierarchy (integer)

OUT `info` info object (handle)

C Prototype :

```
int MPIX_Get_hw_topology_info(int *numlevels,MPI_Info info)
```

Current extension/implementation :
allows the calling MPI process to retrieve information about the underlying hardware topology. Two different types of information are available :

— the number of hardware levels in the hierarchy (the `numlevels` parameter)

— a set of keyvals (set in the `info` parameter). There are *numlevels* different keyvals, named `MPI_HW_LEVEL0`, `MPI_HW_LEVEL1`, ... `MPI_HW_LEVEL`*`numlevels-1`*. These values are implementation-defined and can be the same as the one used to define the `MPI_HW_DOMAIN_TYPE` key in `MPIX_COMM_SPLIT_TYPE`.

2.2 MPIX_GET_HW_DOMAIN_INFO

`MPIX_GET_HW_DOMAIN_INFO(comm,num_subcomms,rank,type,info)`

IN `comm` communicator (handle)

OUT `num_subcomms` number of subcommunicators (integer)

OUT `rank` domain "rank" (integer)

OUT `type` domain name (string)

IN `info` info object (handle)

C Prototype :

```
int MPIX_Get_hw_domain_info(MPI_Comm comm,int *num_subcomms,
                            int *rank,char **type, MPI_Info info)
```

Current extension/implementation :

— Uses internally he MPI_HW_DOMAIN_TYPE, MPI_HW_DOMAIN_NUM, and MPI_HW_DOMAIN_RANK keys defined in MPI_COMM_SPLIT_TYPE, and returns their values to the user in a more usable form.

— Uses an info parameter that should not be in the final version of the interface, as this object should be attached to the input communicator with the MPI_Comm_set_info function in MPI_COMM_SPLIT_TYPE and retrieved with the MPI_Comm_get_info function in MPIX_GET_HW_DOMAIN_INFO.

2.3 MPIX_GET_MIN_HW_DOMAIN

MPIX_GET_MIN_HW_DOMAIN(comm,size,ranks,type)

IN comm communicator (handle)

IN size size of ranks array (integer)

IN ranks ranks (array of integer)

OUT type hardware domain name (string)

C Prototype :

```
int MPIX_Get_min_hw_domain(MPI_Comm comm, int size, int *ranks, char **type)
```

Current extension/implementation :
Returns the name of the *lowest* level in the hierarchy shared by all the MPI processes which ranks in the communicator comm are listed in the ranks array. If the calling process rank is not among the ranks listed in the array passed as an argument, the type returned should be "Unknown" or "Invalid".

3 Code example

```
{
 MPI_Comm out_comm;
 MPI_Info info;
 int rank, scomm_num,scomm_rank;
 char *resource_type = NULL;

 MPI_Info_create(&info);
 MPI_Info_set(info,"MPI_HW_DOMAIN_TYPE","Socket");
 MPI_Comm_rank(MPI_COMM_WORLD,&rank);
 MPIX_Comm_split_type(MPI_COMM_WORLD,MPI_COMM_TYPE_HW_DOMAIN,rank,info,&out_comm);

 /* check info about subcomm */
 if (out_comm != MPI_COMM_NULL){
   MPIX_Get_hw_domain_info(out_comm,&scomm_num,&scomm_rank,&resource_type,info);
   fprintf(stdout,"=== Number of subcomms : %i\n",scomm_num);
   fprintf(stdout,"=== Subcomm rank : %i\n",scomm_rank);
   fprintf(stdout,"=== Subcomm type : %s\n",resource_type);
 } else {
   fprintf(stdout,"No level found\n");
 }
}
```

4 Questions

- Is the extended behaviour of MPI_COMM_SPLIT_TYPE satisfactory (with both the directed and undirected modes) ?
- Can the requirement of producing striclty smaller subcommunicator be enforced through a *specific* key in the info argument (e.g. MPI_HW_SPLIT_SMALLER) ?
- Are M̂PI_COMM_TYPE_HW_DOMAIN and MPI_HW_DOMAIN_TYPE good names ?
- Should *key names* (i.e. MPI_HW_LEVEL0,MPI_HW_LEVEL1 be also usable as *values* for the MPI_HW_DOMAIN_TYPE key ?
- Is MPIX_GET_HW_TOPOLOGY_INFO a good name ?
- how can we improve the prototype fo the function MPIX_GET_HW_TOPOLOGY_INFO ?