# `MPI_Comm_split_type` guided mode query function

Guillaume Mercier

## 1 Context and issue

Issue #132 adds a new `split_type` value for the `MPI_COMM_SPLIT_TYPE` function : `MPI_COMM_TYPE_HW_SUBDOMAIN` in order to create hardware-based communicator that allow th user to better structure their application according the the hadware locality of the processes. The user can specify the hardware resource to perform the splitting operation by providing a value to a new MPI key (`mpi_hw_domain_type`). Such value is an implementation defined string.

The questions that arises are then :

— how can the user know the number of strings available ?

— how can the user know the full list of strings available ?

## 2 Design variations

Several designs are possible to achieve this goal : using the tools interface or adding a new function to the MPI standard.

### 2.1 The Tools interface

Since the strings are implementation defined, the user could leverage the `MPI_T` interface to gain the knowledge of the various strings designating the available resources in the target hardware on which the application is deployed and launched.

=> **No because the MPI_T interface is designed for tools, not applications and it doesn't work with Fortran.**

### 2.2 Adding a specific function to the MPI standard

There are (at least) two variants to retrieve the desired information : one using an info object and one returning an array of strings. => **25/09/2019 Telco :**

— **consensus about using an info object rather that an array of strings**

— **how about adding a comm argument to the routine ?**

**Variant 1 :**

Generic prototype :

```
MPI_GET_HWSUBDOMAIN_NAMES(comm,nresources,info)
```

IN `comm` communicator with hardware info (handle)

IN/OUT `nresources` number of strings (integer)

IN/OUT `info` info argument (handle)

C Prototype :
`int MPI_get_hwsubdomain_names(MPI_Comm comm, int *nresources, MPI_Info info)`
About arguments :

— **the `comm` argument represents a communicator with knowledge of HW info. This must be an intracommunicator**
— the `nresources` parameter is the number of resource as recognized in the target hardware by the MPI implementation
— the `info` object stores a set of keys : there are *nresource* different keys, named `mpi_hw_res_0`, `mpi_hw_res_1`, ... `mpi_hw_res_`*`nresource-1`* . The values are the implementation-defined strings that the user queries.

**Also the names cannot be set by `MPI_Comm_info_set` nor retrieved by `MPI_Comm_info_get`**

Code example :

```
{
 MPI_Comm hwcomm;
 MPI_Comm oldcomm = MPI_COMM_WORLD;
 MPI_Info info;
 char *resource_type = NULL;
 char str[64];
 char *str2 = NULL;
 int rank, idx, flag;
 int num_resource;
 int keylen;

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

 MPI_Get_hwsubdomain_names(&num_resources,info);
 fprintf(stdout,"Number of resource available: %i\n",num_resources);

 for(idx = 0 ; idx < num_resources; idx++){
   sprintf(str,"mpi_hw_res%d",idx);
   MPI_Info_get_valuelen(info,str,&keylen,&flag);
   if(flag){
     str2  = malloc(keylen+1);
     MPI_Info_get(info,str,keylen,str2,&flag);
     if (flag)
       fprintf(stdout,"Resource %s type is: %s\n",str,str2);
   }
 }

 /* Let us suppose that L3Cache is an available resource */
 MPI_Info_set(info,"mpi_hw_domain_type","L3Cache");
```

```
    MPI_Comm_split_type(oldcomm,MPI_COMM_TYPE_HW_SUBDOMAIN,rank,info,&hwcomm);

    /* now use hwcomm ... */
   }
```

**Variant 2 :** The same, but without the first `comm` argument.