`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.

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.

Variant 1 :

Generic prototype :

```
MPI_GET_HWSUBDOMAIN_NAMES(nresource,info)
```

OUT `nresource` number of strings (integer)

OUT `info` info argument (handle)

C Prototype :

```
int  MPI_get_hwsubdomain_names(int *nresource, MPI_Info info)
```

About arguments :

— the `nresource` 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_res0`, `mpi_hw_res1`, ... `mpi_hw_res`*nresource-1*. The values are the implementation-defined strings that the user queries.

Code example :

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

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

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

 for(idx = 0 ; idx < num_resource; idx++){
   sprintf(str,"mpi_hw_res%d",idx);
   MPI_Info_get(info,str,MPI_MAX_INFO_VAL-1,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 :

Generic prototype :

```
 MPI_GET_HWSUBDOMAIN_NAMES(nresource,resource_names)
```

OUT `nresource` number of strings (integer)
OUT names of resources (array of strings)

C Prototype :

```
int  MPI_get_hwsubdomain_names(int *nresource, char **resource_names)
```

This function generalizes the approach taken by `MPI_Get_processor_name`.
About arguments : each string is at most `MPI_MAX_OBJECT_NAME-1` characters (the same as a communicator name).
Question : Is this ok with Fortran ?