IUP MAI3 - TD M54
Topologie d'hypercube en MPI

Notre objectif est de construire une topologie d'hypercube dans un réseau de processus avec MPI. Nous utiliserons la fonction `MPI_Graph_create` de MPI :
`MPI_Graph_create(MPI_Comm CommAncien,int nbnoeuds, int *index, int *aretes,int reorder, MPI_Comm *CommGraph).`
Avec

- `nbnoeuds` est le nombre de noeuds dans le graphe (en général, le nombre de processus `size` dans `CommAncien`).
- `index` est un tableau de dimension `nbnoeuds`. Il permet d'identifier dans le tableau `aretes` les arêtes de chaque sommet, de la manière suivante :
  1. la liste des voisins du processus 0 sont stockés dans `aretes[j]` pour $j = 0, \ldots, index[0] - 1$.
  2. la liste des voisins du processus $i > 0$ sont stockés dans `aretes[j]` pour $j = index[i-1], \ldots, index[i] - 1$.

  En particulier, `(index[i]-index[i-1])` est le nombre de voisins du sommet de rang `i`.
- `aretes` est une liste de noeuds permettant de coder les arêtes du graphe : pour $j = index[i-1], \ldots, index[i] - 1$, il y a une arête entre le noeud de rang i et le noeud de rang `aretes[j]`. La dimension du tableau `aretes` est donc deux fois le nombre d'arêtes du graphe.
- `reorder` est en général mis à 0 (faux), car on ne souhaite par réordonner les processus dans le nouveau communicateur.

Soit `rang` le numéro d'un processus de MPI. Nous rappelons la topologie d'hypercube de dimension p sur un réseau de taille 2^p par : soit $j_{p-1}, j_{p-2}, \ldots, j_1, j_0$ la numérotation en base deux de `rang` , c'est à dire $\forall l = 0, \ldots, p-1, j_l \in \{0, 1\}$ et

$$\texttt{rang} = \sum_{l=0}^{p-1} j_l 2^l,$$

les processus `rangj` $= \sum_{l=0}^{p-1} j_l 2^l$ et `rangk` $= \sum_{l=0}^{p-1} k_l 2^l$ sont voisins ssi $\exists! l_0 \in \{0, \ldots, p-1\}, j_{l_0} \neq k_{l_0}$ (et $\forall l = 0, \ldots, p-1, l \neq l_0 \Rightarrow j_l = k_l$).

1. Construisez le contenu des tableaux `index` et `aretes`, pour les hypercubes de dimension p pour $p = 1, 2, 3, 4$. Prenez soin de conserver l'ordre des directions dans les voisinages, cad, le premier voisin est celui dans la première direction, le second est celui dans la seconde direction, le p^{iem} est celui dans la p^{ieme} direction.

2. Donnez un algorithme qui permet de construire une topologie d'hypercube avec `MPI_Graph_create`, conservant l'ordre des directions dans l'ordre des voisins.

3. En réutilisant l'idée de l'algorithme papillon, proposez un algorithme parallèle qui, en p étapes, diffuse progressivement les données d'un vecteur de taille $N = 2^n$ sur un hypercube de dimension p, en supposant qu'à l'initialisation le processeur 0 possède entièrement le vecteur f et qu'en sortie le processeur `rang` possède $N/2^p$ composantes de f sur lesquels il devra faire une FFT séquentielle de taille 2^{n-p} dans l'agorithme de FFT.

4. En déduire un algorithme distribué de FFT de taille 2^n sur un réseau de processeurs de taille 2^p (avec $p < n$)

Des fonctions utiles :

- `MPI_Graph_neighbors(MPI_Comm CommGraph, int rang, int nbvoisins, int *voisins)` retourne la liste des voisins de `rang` dans le tableau `voisins`.
- `MPI_Graph_neighbors_count(MPI_Comm CommGraph, int rang, int *nbvoisins)` retourne le nombre de voisins de `rang` .