Version MPI du TSP

Il s'agit de programmer une version distribuée du voyageur de commerce à l'aide d'un simple paradigme maître/esclave, où la machine de rang 0 (le « nœud 0 ») distribue du travail aux autres machines (les autres « nœuds » lorsque celles-ci deviennent inoccupées.
Rappel : on pourra bien sûr garder sous les yeux les pages de man de ces fonctions MPI, mais sur `http://mpi.deino.net/mpi/mpi_functions/` la documentation est plus fournie et d'autres exemples de codes sont disponibles. Une documentation très détaillée est disponible sur `http://www.netlib.org/utk/papers/mpi-book/mpi-book.html` .

1 Avant toutes choses

Si vous ne l'avez pas fait la semaine dernière, copiez une clé ssh comme indiqué dans le TP précédent. Modifier à votre convenance le fichier mymachine.

2 Version mono-thread

Pour l'instant faisons simple : un seul thread par machine, et un modèle maître/esclave.
Le nœud 0 s'occupe d'abord de faire le parcours partiel pour remplir sa queue de jobs. Remplissez maintenant sa boucle `while()` pour qu'il passe son temps à attendre un message d'un quelconque autre nœud et lui envoie un chemin partiel. Vous pouvez pour cela utiliser `MPI_Recv` avec la constante `MPI_ANY_SOURCE}`, et savoir le numéro de nœud qui a envoyé le message en consultant `status.MPI_SOURCE` . Pour les données, on utilisera bien sûr le type `MPI_INT` . On utilisera un tag différent pour les demandes et pour les réponses (c'est important pour la fin du TP).
Les autres nœuds, eux, devront passer leur temps à envoyer un message au nœud 0, réceptionner un chemin partiel et le calculer.
Pour la terminaison, on pourra utiliser un message spécial.
Utilisez `make run` pour tester, le fichier `mymachines` inclut de base `localhost` pour utiliser votre propre machine plusieurs fois. Une fois que votre programme fonctionne, alternez dans le fichier `mymachines` entre `localhost` et le nom d'une autre machine, et augmentez la valeur données à l'option `np` dans le `Makefile`. Observez que les différents cœurs sont utilisés. On peut effectivement exploiter les cœurs avec seulement le paradigme MPI !

3 Gestion du min

La variable `minimum` est pour l'instant privée à chaque thread. Comment la rendre *un peu* plus publique pour améliorer les coupures ?

4 Avec des threads

L'implémentation MPI dont vous disposez n'est pas complètement multithreadée, c'est-à-dire que l'application ne peut comporter des threads que si seul le thread principal exécute des appels aux fonctions MPI. Pour pouvoir lancer des threads au sein d'un même nœud, il faudra donc y gérer la distribution du travail !
Combien de nœuds vaut-il mieux lancer maintenant ? Quelle autre solution est possible ?
Obtenez-vous un meilleur speedup qu'avec la solution de facilité de la section précédente ?