Utilisations de processus légers

Résumé: Dans ce TD, nous améliorons notre mise en œuvre de l'algorithme de double diffusion en utilisant des processus légers.

Un des problèmes de notre mise en œuvre précédente est que les communications et les calculs sont complètement séquentiels alors qu'il est parfaitement possible de les faire se recouvrir. On peut par exemple parfaitement effectuer les calculs de la $k^{\text{ème}}$ étape pendant que l'on diffuse les données de la $(k+1)^{\text{ème}}$ étape... Nous allons effectuer ce recouvrement à l'aide de threads. Une des difficultés provient cependant du fait que MPI n'est pas réentrant pour les threads.

1 Quelques informations sur les threads

Pour créer un thread, il suffit d'utiliser la fonction suivante :

```
int pthread_create(pthread_t * thread, pthread_attr_t * attr, void *
                   (*start_routine)(void *), void * arg);
```

Le premier est une structure que vous n'avez pas à utiliser par la suite, le second n'a pas d'importance pour nous (`NULL` ira donc très bien), le troisième est le nom de la fonction que le thread va exécuter et le dernier est un argument qui sera passé en argument à cette fonction. Il est donc courant de se définir un type `struct` en début de programme, d'utiliser une variable de ce type pour stocker les différents arguments de la fonction et de passer un pointeur sur cette structure.

Pour gérer la synchronisation de nos threads, nous allons utiliser des sémaphores. On utilisera la fonction suivante pour créer un sémaphore :

```
int sem_init (sem_t *SEM, int PSHARED, unsigned int VALUE)
```

`SEM` est le sémaphore que l'on souhaite initialiser, `PSHARED` vaudra 0 pour nous et `VALUE` est la valeur initiale du sémaphore.

Les threads peuvent effectuer les opérations suivantes sur un sémaphore :

```
int sem_wait (sem_t * SEM)
```

Cette fonction suspend le thread qui l'appelle jusqu'à ce que la valeur du sémaphore soit strictement positive. Cette valeur est alors atomiquement décrémentée.

```
int sem_post (sem_t * SEM)
```

Cette fonction incrémente atomiquement la valeur du sémaphore et libère donc les éventuels threads qui étaient bloqués. Cette fonction n'est donc jamais bloquante.

2 Mise en œuvre en MPI de l'algorithme par double diffusion avec recouvrement

Pour recouvrir calculs et communications, nous allons créer deux nouveaux threads :
- le premier sera en charge des calculs : à l'étape k, il calculera le résultat du produit de la $k^{\text{ème}}$ colonne avec la $k^{\text{ème}}$ ligne
- le second sera en charge des communications : à l'étape k, il effectuera les diffusions de la $(k+1)^{\text{ème}}$ colonne et de la $(k+1)^{\text{ème}}$ ligne.

Vous trouverez dans le répertoire `/home/lmarchal/td8-src/` un canevas pour le programme de produit matriciel avec recouvrement (`matmult_threaded.c`). Comme d'habitude, le programme est commenté, il n'y a qu'à remplir les trous et il y a un joli makefile qui compile le tout.

Une fois terminée l'implementation, comparez le temps d'exécution de votre programme avec la version sans recouvrement fournie (`matmult.c`), ou celle que vous aviez écrite au TP précédent.