Projet Algorithmique et Programmation Parallèle

Programmation hybride MPI/pthread

Mettez votre travail dans un répertoire NOM_PRENOM/
Il doit contenir le listing de vos sources mais aussi un document pdf expliquant brièvement vos implémentations aux différentes questions (le document ne doit pas dépasser 3-4 pages).
Puis archivez ce répertoire en prenant soin de supprimer tous vos exécutables pour économiser de la place : tar cvfz NOM_PRENOM.tgz NOM_PRENOM/
Envoyez le fichier ***NOM_PRENOM.tgz*** *à david.dureau@cea.fr avant le* ***25/04/2021*** *minuit.*

Mémo : Utilisation MPI en contexte multithread

La programmation hybride consiste à travailler avec des threads POSIX à l'intérieur des processus MPI.
Pour ce faire, l'appel à `MPI_Init` est remplacé par un appel à `MPI_Init_thread`.

```
Int mpi_thread_provided;
MPI_Init_thread(&argc, &argv, MPI_THREAD_SERIALIZED, &mpi_thread_provided);
```

Cet appel initialise un calcul MPI en mode `MPI_THREAD_SERIALIZED` : un appel MPI ne peut être fait que par un seul thread à la fois.

D'autres modes existent mais ne seront pas abordés (`MPI_THREAD_SINGLE, MPI_THREAD_FUNNELED, MPI_THREAD_MULTIPLE`).
Dans la suite, nous utiliserons uniquement le mode `MPI_THREAD_SERIALIZED`.

Problème

Notre objectif est de paralléliser l'algorithme dit du Gradient Conjugué en contexte hybride MPI/pthread.

Dans le répertoire `hybrid_mpi_pthread/`, nous avons les sous-répertoires :

- `mpi_decomp/` : Décomposition MPI en sous-domaine d'un intervalle [0, ntot[, à compiler
- `thr_decomp/` : Décomposition MT en sous-domaine d'un intervalle [0, ntot[, à compiler
- `hyb_reduc/` : Réduction somme hybride MPI/pthread (à completer) avec programme de test
- `hyb_exchg/` : Echanges point à point en contexte hybride entre sous-domaines MPI (à completer) avec programme de test
- `hyb_grad_conj/` : Algorithme de Gradient Conjugué à paralléliser en mode hybride (à

faire)

Réduction somme hybride MPI/pthread

Aller dans le répertoire `hyb_reduc/`.
Objectif : écrire la fonction (dans le fichier `hyb_reduc.c`)

```
void hyb_reduc_sum(double *in, double *out, shared_reduc_t *sh_red)
```

qui est appelée par tous les threads « workers » de tous les processus MPI .
Cette fonction effectue la somme de toutes les contributions locales de tous les threads inter-processus (tableau `in` pour chaque contribution locale) et qui retourne les résultats dans le tableau out (un tableau `out` par thread « worker »).

Pour ce faire, vous devez :

- compléter la structure `shared_reduc_t` (fichier `hyb_reduc.h`)
- implémenter les fonctions `shared_reduc_init` et `shared_reduc_destroy` (fichier `hyb_reduc.c`).

Tester vos fonctions sur le programme `test_hyb_reduc.c` (chaque processus MPI crée `NUM_WORKERS` threads)

Echanges point à point hybrides MPI/pthread

Aller dans le réperoire `hyb_exchg/`.
Objectif : écrire la fonction (dans le fichier `hyb_exchg.c`)

```
void hyb_exchg(
      double *sh_arr,
      shared_exchg_t *sh_ex,
      double *val_to_rcv_left, double *val_to_rcv_right,
      mpi_decomp_t *mpi_decomp)
```

qui est appelée par tous les threads « workers » de tous les processus MPI.
Le tableau `sh_arr` est un tableau distribué selon la décomposition MPI mais partagé parmi les threads « workers » d'un processus.
Si processus MPI existe "à gauche", lui envoie la valeur `sh_arr[0]` et reçoit de lui `*val_to_rcv_left`.
Si processus MPI existe "à droite", lui envoie la valeur `sh_arr[mpi_decomp->mpi_nloc-1]` et reçoit de lui `*val_to_rcv_right`.
Si processus voisin n'existe pas, valeur correspondante affectée à 0.

Pour ce faire, vous devez :

- compléter la structure `shared_exchg_t` (fichier `hyb_exchg.h`)
- implémenter les fonctions `shared_exchg_init` et `shared_exchg_destroy` (fichier `hyb_exchg.c`).

Tester vos fonctions sur le programme `test_hyb_exchg.c`.

Algorithme du Gradient Conjugué parallèle MPI/pthread

Quelques mots sur l'algorithme du Gradient Conjugué (GC):
il s'agit d'un algorithme itératif de résolution de système linéaire de la forme :

$$A.x = b$$

où,

- A est une matrice symétrique définie positive de dimension N,
- x et b sont des vecteurs de dimension N, x est l'inconnue.

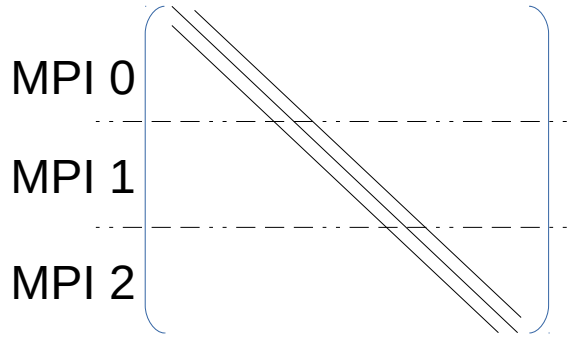
L'algo GC fait notamment appel à des produits matrice-vecteur et à des produits scalaires entre vecteurs.

Description du programme fourni :
Dans le répertoire `hyb_grad_conj/`, le fichier `seq_grad_conj.c` est un programme séquentiel qui :

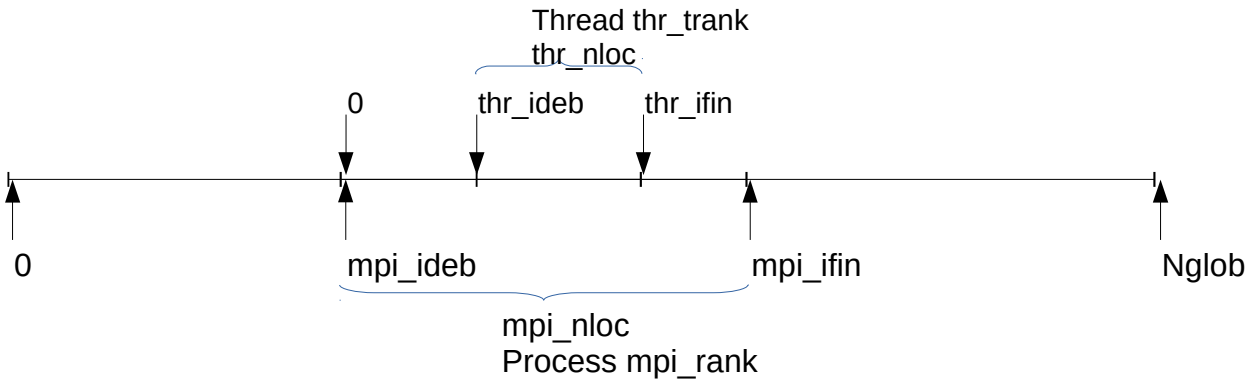
- récupère la dimension du système N en argument ;
- définit la notion de vecteur par le type `vector_t` ainsi qu'une suite d'opérations sur les vecteurs ;
- fait l'hypothèse que A est une matrice creuse à 3 bandes (de type `matrix3b_t`) : pour la ligne i (1 < i < N-1), seuls les éléments correspondant à A(i, i-1), A(i, i) et A(i, i+1) sont considérés comme non nuls et sont stockés (attention aux cas particuliers i = 0 et i = N-1) ;
- construit le système linéaire c'est-à-dire remplit la matrice A et le vecteur b (fonction `linear_system_alloc_and_init`) ;
- résoud le système en appelant l'algorithme du Gradient Conjugué (fonction `gradient_conjugue`)
- vérifie le résultat (fonction `verif_sol`).

Travail à effectuer :
Paralléliser ce programme en contexte hybride MPI/pthread.
La construction du système sera faite par le thread principal.
Puis, chaque processus MPI créera `NUM_WORKERS` threads. Chaque thread « worker » appelera l'algorithme de GC sur sa partie de système puis la fonction de vérification du résultat.

Remarque : pour ce faire, utiliser les types utilisés précédemment à savoir `mpi_decomp_t`, `thr_decomp_t`, `shared_reduc_t` et `shared_exchg_t`.



Une matrice 3-bande distribuée sur 3 processus MPI



Chaque processus MPI s'occupe des lignes [mpi_ideb, mpi_ifin[de la matrice. Chaque processus MPI a sa propre numérotation qui commence à 0. Pour un processus donné, chaque thread s'occupe de son propre intervalle [thr_deb, thr_ifin[.