ESI
Contrôle Intermédiaire semestre I
Documents autorisés.

HPC (2CSSILG1,G2 ; 2CSSIQ4)
Durée : 2 heures
Mercredi 18/11/2018

Exercice 1 : LES THREADS POSIX (10 POINTS).

Nous considérons une matrice de dimension MxM, dont les lignes seront traitées par N threads avec (N < M). Chaque ligne de la matrice n'est traitée qu'**une seule fois.**

Le thread **main** appelle la fonction **create_threads** qui crée *N* threads exécutant la fonction **traiter_lignes**. Un thread **traiter_lignes** reçoit comme argument son numéro d'ordre *i* (*i = 1 .. N*). Il récupère alors la prochaine ligne de la matrice qui n'a pas encore été traitée et la traite. Dès qu'il a fini le traitement, le thread **traiter_lignes** récupère une nouvelle ligne, s'il en reste encore.

Le traitement fait par un thread **traiter_lignes** se résume à multiplier la ligne récupérée par son numéro d'ordre. À la fin, le thread **traiter_lignes** renvoie le nombre de lignes qu'il a traitées.

Le thread **main** attend la fin de tous les threads en appelant la fonction **attendre_fin_traitement** avant d'afficher le contenu final de la matrice *mat*. Cette fonction attend la fin de tous les threads en affichant le nombre de lignes traitées par chaque thread.

Le squelette du programme est donné ci-dessous. La fonction **initialiser_mat** initialise tous les éléments de la matrice **mat** à 1. La fonction **afficher_mat** affiche le contenu de la matrice.

```
#define N 3 /* nombre de threads */
#define M 10 /* dimensions de la matrice */
int mat[M][M];
pthread_t tid [N]; /* sauvegarder les identifiants des threads */
void init_mat (void) {
  int i,j ;
    for (i=0; i< M; i++)
      for (j=0; j< M; j++)
          mat[i][j] =1;
}
void afficher_mat (void) {
  int i,j ;
    for (i=0; i< M; i++) {
       for (j=0; j< M; j++)
         printf ("%d ", mat[i][j]);
         printf ("\n");
    }
}
void *traiter_lignes (void* arg) {
/* a compléter */
}
int create_threads (int nb_threads, pthread_t * tid) {
/* a compléter */
}
int attendre_fin_traitement (int nb_threads, pthread_t * tid) {
/* a compléter */
}
int main (int argc, char ** argv) {
      init_mat ();         /* initialiser matrice */
      if (create_threads (N, tid) == -1) {
            printf ("ERREUR: création threads\n");
            return EXIT_FAILURE;
      }
      if   (attendre_fin_traitement (N, tid) == -1) {
            printf ("ERREUR: fin traitement \n");
            return EXIT_FAILURE;
      }
      afficher_mat ();         /* afficher le contenu final de la matrice */
      return EXIT_SUCCESS;
}
```

ESI
Contrôle Intermédiaire semestre I
Documents autorisés.

HPC (2CSSILG1,G2 ; 2CSSIQ4)
Durée : 2 heures
Mercredi 18/11/2018

Question 1 :

Complétez la fonction *create_threads*. Elle doit créer les *N* threads *traiter_lignes* **joignables** en passant comme argument à chaque thread son numéro d'ordre de création *i* (*i = 1 .. N*).
La fonction reçoit comme paramètres le nombre de threads *nb_threads* à créer et l'adresse de la variable *tid* où les identifiants des threads doivent être sauvegardés. Elle renvoie -1 s'il y a eu un problème lors de la création d'un thread ou 0 en cas de succès.

Question 2 :

Programmez le corps de la fonction *traiter_lignes*. Spécifiez également la déclaration et l'initialisation des variables globales dont vous avez besoin dans votre solution.

Question 3 :

Complétez la fonction *attendre_fin_traitement* qui attend la terminaison de tous les threads *traiter_lignes* en affichant le nombre de lignes traitées par chacun des *threads* ainsi que son numéro d'ordre.
Le nombre de threads à attendre et leur identifiant respectif sont passés en argument. La fonction renvoie -1 s'il y a eu un problème pour attendre la terminaison d'un thread ou 0 en cas de succès.

Question 4 :

Nous voulons maintenant que les threads *traiter_lignes* soient **détachés**. Modifiez la fonction *create_threads* en conséquence.

Question 5 :

Étant détachée, un thread *traiter_lignes* n'informe plus le thread *main* du nombre de lignes qu'il a traitées. Par conséquent, la fonction *attendre_fin_traitement* n'affichera plus cette information. Cependant, le thread *main* doit attendre que toutes les lignes de la matrice soient traitées avant d'afficher le contenu final de la matrice. Modifiez les fonctions *traiter_lignes* et *attendre_fin_traitement* en conséquence.
Observation : Comme dans la question 2, spécifiez la déclaration et l'initialisation des variables globales que vous utilisez dans votre solution.

Exercice 2 : Fonctions sur des vecteurs (OpenMP/C) (5 pts).

Des opérations souvent utilisées lorsqu'on manipule des vecteurs de flottants sont le *SAXPY* (*Single-precision A.X Plus Y*) et le produit scalaire. `saxpy` reçoit un scalaire `a` et deux vecteurs `x` et `y` (de taille `n`), multiplie chaque élément `x[i]` par `a` puis ajoute le résultat à `y[i]`. `produit_scalaire` est utilisé, notamment, pour combiner une ligne (x) et une colonne (y) lors d'un produit matriciel.
On veut paralléliser, en OpenMP/C, les fonctions ci-bas. Complétez-les pour **les rendre les plus parallèles et efficaces possible. Indiquez aussi quelle `schedule` il faudrait utiliser!**

```
void saxpy( float a, float x[], float y[], int n )
{
      for( int i = 0; i < n; i++ ) {
            y[i] = a * x[i] + y[i];
      }
}
float produit_scalaire( float x[], float y[], int n )
{
      float ps = 0.0;
      for( int i = 0; i < n; i++ ) {
            ps += x[i] * y[i];
      }
      return ps;
}
```