UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

LICENCE Informatique
Info0604 – 2019/2020

Travaux dirigés n° 2

Modèles "équipe de travail" et "client-serveur"

Voici un programme `C` implémentant l'addition de 2 matrices :

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NB_LIGNES 6
#define NB_COL 6

int **m1;
int **m2;
int **m3;

void afficherMatrices() {
  int i, j, k;
  int **tabM[3];

  tabM[0] = m1;
  tabM[1] = m2;
  tabM[2] = m3;
  for (k = 0; k < 3; k++) {
    printf("Matrice %d********\n", k+1);
    for (i = 0; i < NB_LIGNES; i++){
      for (j = 0; j < NB_COL; j++)
        printf("%2d ", tabM[k][i][j]);
      printf("\n");
    }
    printf("*****************\n");
  }
}

void initialiserMatricesAlea() {
  int i, j;

  m1 = (int **) malloc(sizeof(int *) * NB_LIGNES);
  m2 = (int **) malloc(sizeof(int *) * NB_LIGNES);
  m3 = (int **) malloc(sizeof(int *) * NB_LIGNES);
  for (i = 0; i < NB_LIGNES; i++) {
    m1[i] = (int *) malloc(sizeof(int) * NB_COL);
    m2[i] = (int *) malloc(sizeof(int) * NB_COL);
    m3[i] = (int *) malloc(sizeof(int) * NB_COL);
    for (j = 0; j < NB_COL; j++) {
      m1[i][j] = (rand() % 5) + 1;
      m2[i][j] = (rand() % 5) + 1;
      m3[i][j] = 0;
    }
  }
}

void additionnerMatrices() {
  int i, j;

  for (i = 0; i < NB_LIGNES; i++){
    for (j = 0; j < NB_COL; j++)
      m3[i][j] = m1[i][j] + m2[i][j];
  }
}

int main(int argc, char *argv[]) {
  srand(time(NULL));
  initialiserMatricesAlea();
  additionnerMatrices();
  afficherMatrices();

  /*liberation memoire des matrices...*/

  return 0;
}
```

On cherche à réduire le temps d'exécution de ce programme en utilisant plusieurs threads qui seront exécutés sur une architecture multiprocesseurs/multicoeurs. Bien que, par souci de simplicité, ce programme se limite à des matrices de 6 lignes et 6 colonnes, il faut bien sur pousser la réflexion en fonction d'une augmentation considérable de la taille des matrices.

1°) Pour chaque fonction prise individuellement, est-il possible d'en dégager du parallélisme ? Pour chaque fonction, dites si c'est possible et proposez différentes façons qui vous semblent pertinentes.

2°) Proposez une implémentation `C/pthreads` où chaque fonction parallélisable, prise individuellement, effectue ses calculs en attribuant une ligne de chaque matrice à un thread différent. Récrivez le main du programme en conséquence.

3°) Si nous choisissons d'exécuter indépendamment chaque procédure d'initialisation/addition en utilisant un thread par ligne, est-il nécessaire de synchroniser le début et la fin de chaque thread pour une ligne donnée ? En fonction de votre réponse, proposez une amélioration à votre programme.

4°) Au niveau système, quel impact aura l'exécution du programme développé précédemment si nous traitons de très grandes matrices ? Peut-on améliorer davantage le temps d'exécution de notre programme ?

Supposons maintenant que (pour une raison obscure) vous voulez permettre à une autre équipe de threads d'effectuer des modifications aléatoires sur les matrices. Chaque thread, à un intervalle de temps variable, modifie un élément spécifique dont les coordonnées et la valeur sont choisis au hasard. Il devra alors modifier toutes les matrices en conséquence. Voici la routine de thread en question :

```
void *threadModifElements(void *arg) {
  int i , j, tempsSleep;

  while (1) {
    tempsSleep = rand() % 3;
    sleep(tempsSleep);
    i = rand() % NB_LIGNES;
    j = rand() % NB_COL;
    m1[i][j] = (rand() % 5) + 1;
    m2[i][j] = (rand() % 5) + 1;
    m3[i][j] = m1[i][j] + m2[i][j];
  }
  return NULL;
}
```

Dans votre programme, cette routine sera exécutée par plusieurs threads :

```
for (i = 0; i < MAX_THREADS; i++)
  statut = pthread_create(&threads[i], NULL, threadModifElements, NULL);
}
```

5°) Comment peut-on protéger les matrices des mises à jour concurrentes ? Proposez une solution où l'on protège chaque position individuelle des matrices.

6°) Cette solution vous semble-t-elle convenable ? Si oui, pourquoi ? Si non, comment améliorer la situation ? Modifiez votre programme en conséquence.

Comme les matrices sont maintenant modifiées régulièrement, on veut pouvoir les afficher à intervalles réguliers. Jusqu'à présent, l'affichage était réalisé une seule fois à la fin du programme. Nous voulons maintenant permettre l'affichage « sur demande » par un thread d'affichage.

7°) Modifiez la routine d'affichage pour qu'elle fonctionne en "mode serveur". Modifiez votre programme pour que l'affichage initial soit effectué en "mode client".

Comme les matrices sont maintenant modifiées à différents intervalles, nous voulons rafraîchir l'affichage de façon régulière, de sorte à suivre l'évolution des valeurs des matrices dans le temps. Il faut toutefois s'assurer que les matrices ne sont pas modifiées en même temps qu'il y a affichage.

8°) Modifiez les routines d'affichage et de mise à jour des matrices de façon à tenir compte de ces contraintes. Utilisez un compteur qui sera incrémenté à chaque modification et qui déclenchera l'affichage lorsque le compteur aura atteint une certaine valeur.