Université de Reims Champagne-Ardenne
Département de Mathématiques, Mécanique et Informatique

Travaux pratiques 3

Modèles « équipe de travail » et « client-serveur »

Voici un programme C implémentant la multiplication séquentielle d'une matrice avec un vecteur :

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

#define M          6
#define N          6

int **matrice;
int *vecteur;
int *vecResultat;

void afficherMatrice(int** matrice, int m, int n) {
   int i, j;
   printf("Matrice******\n");
   for (i = 0; i < m; i++){
      for (j = 0; j < n; j++)
         printf("%d ", matrice[i][j]);
      printf("\n");
   }
   printf("**************\n");
}

void afficherVecteur(int* vecteur, int taille) {
   int i;
   printf("Vecteur******\n");
   for (i = 0; i < taille; i++)
      printf("%d ", vecteur[i]);
   printf("\n");
   printf("*************\n");
}
```

```
int main(int argc, char *argv[]) {
   int i, j;

   srand(time(NULL));

   matrice = (int **) malloc(sizeof(int *) * M);
   vecteur = (int *) malloc(sizeof(int) * N);
   for (i = 0; i < M; i++) {
      matrice[i] = (int *) malloc(sizeof(int) * N);
      for (j = 0; j < N; j++)
         matrice[i][j] = (rand() % 5) + 1;
   }
   for (i = 0; i < N; i++)
      vecteur[i] = (rand() % 5) + 1;

   vecResultat = (int *) malloc(sizeof(int) * M);
   for (i = 0; i < M; i++)
      vecResultat[i] = -1;

   afficherMatrice(matrice, M, N);
   afficherVecteur(vecteur, N);

   for (i = 0; i < M; i++)       {
      vecResultat[i] = 0;
      for (j = 0; j < N; j++)
         vecResultat[i] = vecResultat[i] + matrice[i][j] * vecteur[j];
   }

   afficherVecteur(vecResultat, M);

   free(vecteur);
   free(vecResultat);
   for (i = 0; i < M; i++)
      free(matrice[i]);
   free(matrice);

   return 0;
}
```

Vous pouvez télécharger le code du programme **mat_vec.c** sur la page Moodle du cours.

L'objectif du TP est de développer une version parallèle multithreadée de ce programme en utilisant la bibliothèque pthreads. La stratégie proposée est de répartir les lignes de la matrice sur les threads disponibles.

Étape 1 : un thread par ligne

1. Écrivez une routine de thread permettant le calcul de la valeur d'indice *i* du vecteur résultat. Pour ce faire, vous aurez besoin de la ligne i de la matrice et de tout le vecteur. Modifiez ensuite le programme pour que *m* threads effectuent le calcul du vecteur résultat. Vérifiez le résultat.

Étape 2 : un thread pour plusieurs lignes

2. Modifiez votre programme de sorte à ce que chaque thread calcule *m*/*p* éléments contigus du vecteur résultat. Chaque thread recevra donc l'indice de début du bloc qu'il aura à traiter. Pour simplifier, vous pouvez supposer que le nombre d'éléments du vecteur résultat est un multiple du nombre de threads. Vérifiez le résultat.

Étape 3 : affichage progressif des éléments du vecteur avec ncurses

Plutôt que d'afficher le vecteur résultat complet à la toute fin du calcul, on veut plutôt l'afficher dans une fenêtre ncurses à mesure que ses éléments sont calculés. Un thread spécifique sera alors chargé de mettre à jour l'affichage de l'état actuel du vecteur dans une interface fenêtrée : lorsque *x* (défini en constante ou passé en paramètre au programme) éléments du vecteur résultat sont calculés, le thread d'affichage actualise l'interface.

Quelques éléments à considérer :

- On ne veut pas afficher des valeurs « partielles » des éléments du vecteur résultat : soit on affiche -1 si l'élément n'a pas encore été calculé, soit la valeur si l'élément a été calculé ;
- On veut minimiser la perte de performance associée à la gestion de l'affichage (verrous et attentes).

3. Modifiez votre programme pour que l'affichage soit géré par une interface graphique fenêtrée ncurses.

4. Créez un thread d'affichage qui actualisera l'interface fenêtrée à intervalles réguliers et modifiez votre programme en conséquence. Vérifiez le résultat de votre programme lorsque vous demandez l'affichage pour des valeurs de *x* variant entre 1 et *m*.