Oumy GAYE P28 2551 et Kossivi GNOZIGUE P33 6569

Exercice 19 : CHIMIE COMPUTATIONNELLE.

Résumé en Chimie : INDICATION DU TRAVAIL A FAIRE
Nous avons développé deux algorithmes parallèles pour la construction de la matrice de Fock.

1. Première approche : réplication totale
○ Les matrices F et D sont répliquées dans chacune des N tâches.
○ Les calculs d'intégrales sont répartis entre ces tâches.
○ Un algorithme de sommation est utilisé pour additionner les contributions à la matrice F accumulées dans les différentes tâches.
○ Cet algorithme est simple, mais non évolutif (non scalable).

2. Deuxième approche : réplication partielle et agglomération
○ Les matrices F, D et A sont partagées entre les N tâches, avec une petite quantité de réplication.
○ Les calculs d'intégrales sont agglomérés en un certain nombre de tâches de calcul, chacune contenant plusieurs intégrales.
○ Ces tâches sont ensuite affectées aux processeurs, soit de manière statique, soit via un ordonnancement dynamique.

Analyse des compromis : Cette étude de cas met en lumière certains arbitrages dans la conception des algorithmes :
● Le premier algorithme réduit drastiquement les coûts de communication et de développement logiciel, mais n'est pas évolutif.
● Le second algorithme a un coût de communication plus élevé, mais il est très évolutif : ses besoins en mémoire augmentent uniquement avec la taille du problème, et non avec le nombre de processeurs. Pour choisir entre ces deux algorithmes, il est nécessaire de quantifier leurs performances parallèles et d'évaluer l'importance de l'évolutivité, en fonction des exigences de l'application et des caractéristiques de l'ordinateur parallèle cible.

REPLICATION TOTALE

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <mpi.h>
#include<time.h>
#define N 100
void init_matrix(double* mat, int size) {
   for (int i = 0; i < size; i++) {
      mat[i] = (double) rand() / RAND_MAX;
   }
}

int main(int argc, char** argv) {
   int rank, size;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);

   srand(time(NULL) + rank); // Seed unique par processus

   double *D = malloc(N * N * sizeof(double));
   double *A = malloc(N * N * N * N * sizeof(double)); // Simplification en 1D
   double *F_local = calloc(N * N, sizeof(double));
   double *F_total = NULL;

   if (rank == 0) {
      init_matrix(D, N);
      for (int i = 0; i < N*N*N*N; i++) {
         A[i] = (double) rand() / RAND_MAX;
      }
   }

   MPI_Bcast(D, N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   MPI_Bcast(A, N*N*N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

   int rows_per_proc = N / size;
   int start_row = rank * rows_per_proc;
   int end_row = (rank == size - 1) ? N : start_row + rows_per_proc;

   double start_time = MPI_Wtime();

   for (int i = start_row; i < end_row; i++) {
      for (int j = 0; j < N; j++) {
         double sum = 0.0;
         for (int k = 0; k < N; k++) {
```

```
            for (int l = 0; l < N; l++) {
                int idx = ((i * N + j) * N + k) * N + l;
                sum += D[k * N + l] * A[idx];
            }
        }
        F_local[i * N + j] = sum;
    }
  }

  if (rank == 0) F_total = calloc(N * N, sizeof(double));
  MPI_Reduce(F_local, F_total, N * N, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

  double end_time = MPI_Wtime();

  if (rank == 0) {
    printf("Temps d'exécution (réplication totale) : %f secondes\n", end_time - start_time);
  }

  free(D); free(A); free(F_local);
  if (rank == 0) free(F_total);

  MPI_Finalize();
  return 0;
}
```

REPLICATION PARTIELLE

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>
#include <stddef.h>
#define N 100 // Taille de la matrice (modifiable)
int main(int argc, char** argv) {
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  srand(time(NULL) + rank);//pour avoir des donnees deffirentes dans chaque processeur
  int  nbre_lign_proc = N / size;
  int debut_lign = rank * nbre_lign_proc;
  int fin_lign = (rank == size - 1) ? N : debut_lign +  nbre_lign_proc;
  double *D_local = malloc(N *  nbre_lign_proc * sizeof(double));//: chaque processus n'a que les
lignes de D qui l'intéressent
  double *A = malloc(N*N*N*N * sizeof(double));
  double *F_local = calloc(N *  nbre_lign_proc, sizeof(double));// résultat local pour les lignes
calculées
```

```
  double *F_total = NULL;
  if (rank == 0) {
    for (int i = 0; i < N*N*N*N; i++) A[i] = (double) rand() / RAND_MAX;
  }
  MPI_Bcast(A, N*N*N*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
  for (int i = 0; i < N *  nbre_lign_proc; i++) {
    D_local[i] = (double) rand() / RAND_MAX;
  }
  double debut = MPI_Wtime();
  for (int a = 0; a<  nbre_lign_proc; a++) {
    int i = debut_lign + a;
    for (int j = 0; j < N; j++) {
      double sum = 0.0;
      for (int k = 0; k < N; k++) {
        for (int l = 0; l < N; l++) {
          int idx = ((i * N + j) * N + k) * N + l;
          sum += D_local[a * N + k] * A[idx];
        }
      }
      F_local[a * N + j] = sum;
    }
  }
  if (rank == 0) F_total = calloc(N * N, sizeof(double));
  MPI_Gather(F_local, N *  nbre_lign_proc, MPI_DOUBLE, F_total, N *  nbre_lign_proc,
MPI_DOUBLE, 0, MPI_COMM_WORLD);
  double Fin = MPI_Wtime();
  if (rank == 0) {
    printf("Temps d'exécution (réplication partielle) : %f secondes\n", Fin- debut);
  }
  free(D_local); free(A); free(F_local);
  if (rank == 0) free(F_total);
  MPI_Finalize();
  return 0;
}
```

RESULTATS APRES EXECUTION :

REPLICATION TOTALE

```
:\Users\bmd tech>docker cp "C:\Users\bmd tech\Documents\exo19totale.c" admiring_booth:/root
uccessfully copied 3.58kB to admiring_booth:/root

:\Users\bmd tech>docker exec -it  admiring_booth bash
oot@8d2c9cb8406b:/# cd /root
oot@8d2c9cb8406b:~# mpicc exo19totale.c -o exo19totale
oot@8d2c9cb8406b:~# export OMPI_ALLOW_RUN_AS_ROOT=1
oot@8d2c9cb8406b:~# export OMPI_ALLOW_RUN_AS_ROOT_CONFIRM=1
oot@8d2c9cb8406b:~# mpirun -np 4 ./exo19totale
emps d'ex◆cution (r◆plication totale) : 1.399904 secondes
oot@8d2c9cb8406b:~# mpirun -np 3 ./exo19totale
emps d'ex◆cution (r◆plication totale) : 0.229497 secondes
oot@8d2c9cb8406b:~# mpirun -np 2 ./exo19totale
emps d'ex◆cution (r◆plication totale) : 0.235195 secondes
oot@8d2c9cb8406b:~# mpirun -np 1 ./exo19totale
emps d'ex◆cution (r◆plication totale) : 0.653269 secondes
oot@8d2c9cb8406b:~# exit
```

REPLICATION PARTIELLE

```
C:\Users\bmd tech>docker exec -it  admiring_booth bash
root@8d2c9cb8406b:/# cd /root
root@8d2c9cb8406b:~# export OMPI_ALLOW_RUN_AS_ROOT=1
root@8d2c9cb8406b:~# export OMPI_ALLOW_RUN_AS_ROOT_CONFIRM=1
root@8d2c9cb8406b:~# mpirun -np 1 ./exo19partiel
Temps d'ex◆cution (r◆plication partielle) : 0.414714 secondes
root@8d2c9cb8406b:~# mpirun -np 2 ./exo19partiel
Temps d'ex◆cution (r◆plication partielle) : 0.375558 secondes
root@8d2c9cb8406b:~# mpirun -np 3 ./exo19partiel
Temps d'ex◆cution (r◆plication partielle) : 0.315476 secondes
root@8d2c9cb8406b:~# mpirun -np 4 ./exo19partiel
Temps d'ex◆cution (r◆plication partielle) : 2.191971 secondes
root@8d2c9cb8406b:~#
```