Oumy GAYE P28 2551 et Kossivi GNOZIGUE P33 6569

Exercice 8. Agglomération et Optimisation de la Concurrence

Cas d'étude : Calcul de la somme d'un grand tableau en parallèle

Instructions :

1. Implémentez une version naïve de la somme parallèle.
2. Optimisez cette version en appliquant l'agglomération et en limitant les communications inutiles.
3. Expérimentez avec différents nombres de tâches et de processeurs.
4. Analysez l'impact sur le temps d'exécution et la communication.

Questions :

1. Quelle est la réduction du nombre de communications après agglomération ?
2. Comment choisir la taille optimale d'une tâche après regroupement ?
3. Comment l'agglomération affecte-t-elle l'évolutivité de l'algorithme ?
4. Peut-on appliquer cette technique à d'autres algorithmes de calcul parallèle ?

IMPLEMENTATION DE LA METHODE NAÏVE

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

#define N 10000000   // Taille du tableau
const MAX_NUMBS = 100; // Valeur maximale des éléments du tableau

int main(int argc, char** argv) {
   int rank, size;
   int* tab = NULL;  // Stocké uniquement par le maître
   int* tab_esclave = NULL;    // Partie traitée par chaque esclave
   int taille_tab_esclave;
   int somme_esclave = 0; // Stocke la somme que chaque esclave a trouvé
   int somme_totale = 0;  // Stocke la somme totale obtenue par le maître
   double temps_debut, temps_fin;   // Variables pr lancer et arrêter le chrono

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);

   // Amorcez le générateur de nombres aléatoires pour obtenir des résultats différents à chaque fois
   srand(time(NULL));

 // Le maître initialise le tableau global
   if (rank == 0) {
      tab = (int*)malloc(N * sizeof(int)); //allocation mémoire
      int i;
      for (i = 0; i < N; i++) {
         tab[i] = (rand()% MAX_NUMBS+1);  //Nbre entre 0 et 100
      }
      temps_debut = MPI_Wtime();

      taille_tab_esclave = N / (size - 1); // Taille des tableaux que chaque esclave doit considérer.
   }

   // Envoie de la taille des sous-tableaux à tous les esclaves par la diffusion
   MPI_Bcast(&taille_tab_esclave, 1, MPI_INT, 0, MPI_COMM_WORLD);

   // Allocation mémoire pour les sous-tableaux (esclaves)
   if (rank != 0) {
      tab_esclave = (int*)malloc(taille_tab_esclave * sizeof(int));
   }

   // Le maître envoie les données aux esclaves
   if (rank == 0) {
      int i;
      for (i = 1; i < size; i++) {
         int debut = (i - 1) * taille_tab_esclave; //Le debut du tableau pour chaque esclave
         // Envoie bloquant des sous tableaux à chaque esclave (communication synchrone)
```

```
            MPI_Send(&tab[debut], taille_tab_esclave, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
    } else {
        // Réception des sous tableaux par chaque esclave
        MPI_Recv(tab_esclave, taille_tab_esclave, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    }

    // Calcul de la somme par chaque esclaves
    if (rank != 0) {
        somme_esclave = 0;
        int i;
        for (i = 0; i < taille_tab_esclave; i++) {
            somme_esclave += tab_esclave[i];
        }
        printf("L'esclave %d a trouvé une somme de : %d\n", rank, somme_esclave);
    }

    // Les esclaves envoient leur résultat au maître
    if (rank != 0) {
        MPI_Send(&somme_esclave, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }

    // Le maître somme les résultats reçus
    if (rank == 0) {
        somme_totale = 0;
        int i;
        for (i = 1; i < size; i++) {
            int temp;
            MPI_Recv(&temp, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            somme_totale += temp;
        }

        // Gestion du reste (si N n'est pas divisible par (size-1))
        int reste = N % (size - 1);
        if (reste != 0) {
            int i;
            for (i = N - reste; i < N; i++) {
                somme_totale += tab[i];
            }
        }

            temps_fin = MPI_Wtime();
        printf("La somme totale est : %d\n", somme_totale);
            printf("Le temps d'exécution est de %f secondes\n", temps_fin - temps_debut);
    }

    // Nettoyer les tableaux
    if (rank == 0) {
        free(tab);
    } else {
        free(tab_esclave);
```

```
    }

    MPI_Finalize();
    return 0;
}
```

OPTIMISATION DU CODE EN APPLIQUANT L'AGGLOMERATION ET EN REDUISANT LA COMMUNICATION

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

#define N 10000000      // Taille du tableau
const MAX_VALUE = 100;   // Valeur maximale des éléments du tableau

int main(int argc, char **argv) {
    int rank, size;
    int *tab = NULL;   // Tableau créé par le maître
    int *tab_esclave = NULL;    // Sous tableau affecté aux esclaves
    int taille_tab_esclave;     // Taille des sous tableau
    int somme_esclave = 0;      // Somme des éléments pour chaque esclave
    int somme_totale = 0;       // Somme totale calculée par le maître
    double temps_debut, temps_fin;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Le maître initialise le tableau global
    if (rank == 0) {
        tab = (int *)malloc(N * sizeof(int));
        srand(time(NULL));
        int i;
        for (i = 0; i < N; i++) {
            tab[i] = rand() % MAX_VALUE + 1;  // générer des nombres aléatoires entre 0 et 100
        }
        temps_debut = MPI_Wtime();
    }

    // Calcul des tailles pour les esclaves et des décalages avec gestion du reste
    int *NbreEnvoi = (int *)malloc(size * sizeof(int));
    int *deplacement = (int *)malloc(size * sizeof(int));
    int reste = N % size;
    int sum = 0;
    int i;
    for (i = 0; i < size; i++) {
        NbreEnvoi[i] = N / size + (i < reste ? 1 : 0); //si i <reste, on ajoute 1 sinon on ajoute 0 à N/size
        deplacement[i] = sum;
        sum += NbreEnvoi[i];
    }
```

```
   taille_tab_esclave = NbreEnvoi[rank];
   tab_esclave = (int *)malloc(taille_tab_esclave * sizeof(int));    // Allocation du sous-tableau local

   // Distribution avec MPI_Scatterv (gère les tailles inégales)
   MPI_Scatterv(tab, NbreEnvoi, deplacement, MPI_INT, tab_esclave, taille_tab_esclave, MPI_INT, 0,
MPI_COMM_WORLD);

   // Somme locale (chaque processus traite son sous-bloc)
    if (rank != 0) {
      somme_esclave = 0;
      int i;
           for (i = 0; i < taille_tab_esclave; i++) {
           somme_esclave += tab_esclave[i];
           }
           printf("L'esclave %d a trouvé une somme de : %d\n", rank, somme_esclave);
           }

   // Réduction en arbre (MPI_Reduce utilise une optimisation similaire)
   MPI_Reduce(&somme_esclave, &somme_totale, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

   // Affichage du résultat (maître)
   if (rank == 0) {
      temps_fin = MPI_Wtime();
      printf("La somme totale est : %d\n", somme_totale);
      printf("Le temps d'exécution est de %f secondes\n", temps_fin - temps_debut);
   }

   // Nettoyer les tableaux
   if (rank == 0) free(tab);
   free(tab_esclave);
   free(NbreEnvoi);
   free(deplacement);

   MPI_Finalize();
   return 0;
}
```

EXPERIENCE AVEC DIFFERENTS NOMBRES DE TACHES ET DE PROCESSEURS

Méthode naïve

```
C:\Users\bmd tech>docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED       STATUS          PORTS     NAMES
8d2c9cb8406b   ubuntu    "/bin/bash"   11 days ago   Up 53 minutes             admiring_booth

C:\Users\bmd tech>docker exec -it admiring_booth bash
root@8d2c9cb8406b:/# cd /root
root@8d2c9cb8406b:~# mpicc EX08.c -o EX08
EX08.c:8:7: warning: type defaults to 'int' in declaration of 'MAX_NUMBS' [-Wimplicit-int]
    8 | const MAX_NUMBS = 100; // Valeur maximale des ♦l♦ments du tableau
      |       ^~~~~~~~~~
root@8d2c9cb8406b:~# export OMPI_ALLOW_RUN_AS_ROOT=1
root@8d2c9cb8406b:~# export OMPI_ALLOW_RUN_AS_ROOT_CONFIRM=1
root@8d2c9cb8406b:~# mpirun -np 4 ./EX08
L'esclave 1 a trouv♦ une somme de : 168265528
L'esclave 2 a trouv♦ une somme de : 168288055
L'esclave 3 a trouv♦ une somme de : 168263922
La somme totale est : 504817562
Le temps d'ex♦cution est de 0.067056 secondes
root@8d2c9cb8406b:~# mpirun -np 3 ./EX08
L'esclave 1 a trouv♦ une somme de : 252584635
La somme totale est : 505070104
Le temps d'ex♦cution est de 0.056480 secondes
L'esclave 2 a trouv♦ une somme de : 252485469
root@8d2c9cb8406b:~# mpirun -np 2 ./EX08
La somme totale est : 504927449
Le temps d'ex♦cution est de 0.106531 secondes
L'esclave 1 a trouv♦ une somme de : 504927449
root@8d2c9cb8406b:~#
```

Méthode optimisée :

```
root@8d2c9cb8406b:~# mpicc EX08_opt.c -o EX08_opt
EX08_opt.c:8:7: warning: type defaults to 'int' in declaration of 'MAX_VALUE' [-Wimplicit-int]
    8 | const MAX_VALUE = 100;   // Valeur maximale des ♦l♦ments du tableau
      |       ^~~~~~~~~
root@8d2c9cb8406b:~# export OMPI_ALLOW_RUN_AS_ROOT=1
root@8d2c9cb8406b:~# export OMPI_ALLOW_RUN_AS_ROOT_CONFIRM=1
root@8d2c9cb8406b:~# mpirun -np 4 ./EX08_opt
L'esclave 1 a trouv♦ une somme de : 126270832
L'esclave 2 a trouv♦ une somme de : 126215070
L'esclave 0 a trouv♦ une somme de : 126238469
L'esclave 3 a trouv♦ une somme de : 126210384
La somme totale est : 504934755
Le temps d'ex♦cution est de 0.054205 secondes
root@8d2c9cb8406b:~# mpirun -np 3 ./EX08_opt
L'esclave 1 a trouv♦ une somme de : 168333555
L'esclave 2 a trouv♦ une somme de : 168345535
L'esclave 0 a trouv♦ une somme de : 168270632
La somme totale est : 504949722
Le temps d'ex♦cution est de 0.057790 secondes
root@8d2c9cb8406b:~# mpirun -np 2 ./EX08_opt
L'esclave 0 a trouv♦ une somme de : 252463631
L'esclave 1 a trouv♦ une somme de : 252550181
La somme totale est : 505013812
Le temps d'ex♦cution est de 0.075373 secondes
root@8d2c9cb8406b:~# mpirun -np 1 ./EX08_opt
L'esclave 0 a trouv♦ une somme de : 505054377
La somme totale est : 505054377
Le temps d'ex♦cution est de 0.072956 secondes
```

ANALYSE DE L'IMPACT SUR LE TEMPS D'EXECUTION ET LA COMMUNICATION

Les temps d'exécution avec la méthode naïve :

Pour 4 processeurs : 0.067056 secondes
Pour 3 processeurs : 0.056480 secondes
Pour 2 processeurs : 0.106431 secondes
Nous remarquons que le temps d'exécution est très grand pour 2 processeurs par rapport aux deux autres temps (cas de 3 et 4 processeurs). En suite le temps d'exécution pour 4 processeurs est plus grand que celui de 3 processeurs. Lorsque nous avons 2 processeurs, il y n'y a qu'un seul esclave qui fait tout le calcul. On pourrait dire que le calcul n'est pas parallélisé raison pour laquelle il y a un long temps d'exécution. Donc la communication n'est pas forcément un facteur limitant. Dans les deux autres cas, le temps d'exécution dépend du type de communication. Plus on a de processeurs, plus le temps est long et ceci peut être dû à la synchronisation entre les processeurs lors de l'envoi et de la réception des données ; c'est-à-dire au type de communication entre les processeurs (MPI_Send, MPI_Recv). Il faut donc une optimisation pour que la communication ne soit plus un facteur limitant.

Les temps d'exécution avec la méthode optimisée :

Pour 4 processeurs : 0.054205 secondes
Pour 3 processeurs : 0.057790 secondes
Pour 2 processeurs : 0.075373 secondes
Pour 1 processeur : 0.072956 secondes
Nous remarquons de façon générale que plus le nombre de processeurs est grand, plus le temps d'exécution est rapide. La communication n'est plus un facteur limitant car elle a été optimisée grâce aux communications collectives de sorte que le parallélisme ne soit plus un problème.

Reponses

1. Le nombre de réduction pour 4 processeurs:
   Pour la méthode naïve, en supposant un envoi non optimisé à travers MPI_Bcast, nous avons 3 communication entre le maître et les esclaves sinon 2 (diffusion en anneau). S'ajoute à cela 6 communication à travers MPI_Send et 6 autres à travers MPI_Recv entre maître et esclaves. Au total il y a 15 communications pour les envois séquentiels des messages de diffusion et 14 sinon.
   Pour la méthode d'agglomération, comme précédemment, le nombre de communication dépend du type de MPI. Pour un MPI optimisé, il y aura 2 communications à travers MPI_Scatterv et 2 à travers MPI_Reduce entre le maître et les esclaves ; ce qui donne 4 au total. Par contre pour un MPI naïf, il y aura une communication séquentielle donc 3 pour chaque fonction ce qui donne un total de 6 communications.
   En définitive, le nombre de réduction dépend du type de MPI. Pour un MPI non séquentiel, il y a une réduction de 10 communications et 9 pour un MPI séquentiel.
2. La taille optimale a été choisie par rapport au nombre de processeurs. Le but est d'affecter approximativement, le même nombre de tâche à chaque processeur.
3. L'agglomération a permis d'améliorer le temps d'exécution en réduisant le nombre de communication, en équilibrant les tâches à réaliser par tous les processeurs y compris le maître.
4. Oui, on peut affecter cet algorithme à d'autres type de programmation parallèles.