RAPPORT POUR LE TP MPI

FACULTÈ DES MATHEMATIQUES ET DE L'INFORMATIQUE

DEPARTEMENT DE L'INFORMATIQUE

OPTION : GENIE INFORMATIQUE

Par : **HANANE MEFTAHI**

1. INTRODUCTION

Les images peuvent parfois être assez volumineuses et leur manipulation peut être gourmande en calculs. Les vidéos aussi.

Récemment, la NASA a téléchargé une image de toute une galaxie ayant N étoiles ,donc le but de ce TP etant d'ecrire et de paralléliser un programme qui compte le nombre d'étoiles dans une image très détaillée et à haute densité de pixels de la galaxie d'Andromède, un énorme fichier JPEG(1.7 GB).

L'une des tâches consistait à trouver la sequence/partie où le programme est réellement parallélisable. L'autre était de faire en sorte que le nombre réel d'étoiles soit aussi proche que les valeurs de la NASA,en utilisant **P**ython **comme langage de programmation ,** MPI **Librairie** et certains outils de traitement d'image comme openCV.

2. CODE SEQUENTIEL

```
import numpy as np
import cv2
import time
tps_deb = time.time()
img = cv2.imread('C:/Users/Hanane/Downloads/heic1502a.tif',0)
img_thresh =  cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,59,0)
nbr_total_etoiles = ((200 < img_thresh)).sum()
print("temps d'execution du programme est:",(time.time()) - tps_deb,"milisecondes")
print("le nombre total des etoiles est : ", nbr_total_etoiles )
```

3. CODE PARALLELE

3.1 ETAPES DU CODE PARALLELE

Voici les etapes effectueès pour paralléliser le code.

Commençons. Une fois que vous avez installé MPI4PY et OpenCV, dans un shell Anaconda, importez les packages suivants.

```
import numpy as np              #  IMPORTER LA BIBLIOTHEQUE 'numpy' COMME 'np'
import cv2                      #
from mpi4py import MPI          # IMPORTER 'MPI' DE LA BIBLIOTHÈQUE 'mpi4py'
import time                     # IMPORTER 'time' MODULE
```

Tout problème MPI aura un communicateur défini. Il s'agit d'un ensemble de processus qui aident à exécuter le code MPI en communiquant entre eux. Ici, nous définissons une communication communicator. La taille et le rang du communicateur sont également définis.

```
comm = MPI.COMM_WORLD           # DÉFINIR LE COMMUNICATEUR
size = comm.Get_size()          # TROUVEZ LA TAILLE DU COMMUNICATOR
rang = comm.Get_rank()          # TROUVEZ LE RANG DU NŒUD ACTUEL
```

Le fichier image est énorme. Notre objectif final est de compter le nombre d'étoiles dans l'image aussi précisément que possible dans le moins de temps possible. Voici le plan :

1. Lisez l'image importèe sur un nœud.
2. Divisez l'image en petits morceaux.
3. Diffusez les dimensions de l'image locale sur chaque nœud.
4. Diffusez chaque partie de l'image sur différents nœuds.
5. Comptez le nombre d'étoiles dans chaque nœud.
6. Calculez le nombre d'étoiles en arrière en utilisant l'opération somme.
7. Affichez le résultat.

- **ETAPE 1, 2:**

Téléchargez l'image sur un nœud. Appelons-le le nœud maître et utilisons rang comme zéro. Donc, uniquement dans le rang == 0 lire l'image téléchargée, enregistrez le temps nécessaire pour lire l'image. Un autre temporisateur est indiqué pour trouver le temps total nécessaire à l'exécution du code. Les

dimensions de l'image entière sont découvertes et les dimensions locales de l'image qui seront dispersées vers d'autres nœuds sont calculées. Dans tous les cas où rang! = 0, nous définissons les variables comme NONE. Les variables doivent être définies sur NONEafin d'éviter les erreurs lors des étapes de BROADCAST et de SCATTER dans les nœuds autres que le nœud maître. Le type de données utilisé pour définir les fichiers image est uint8.

```
if rang == 0:                    # CHARGER L'IMAGE UNIQUEMENT DANS LE NŒUD MAÎTRE
        t1 = MPI.Wtime()         # TROUVEZ LE TEMPS NÉCESSAIRE POUR CHARGER L'IMAGE
        img = cv2.imread ('C:/Users/Hanane/Downloads/heic1502a.tif',0)#CHARGEZ L'IMAGE
        t2 = MPI.Wtime()
        print ("temps pris pour la lecture de l'image est : ", t2-t1)
        # AFFICHER LE TEMPS NÉCESSAIRE POUR CHARGER L'IMAGE
        nbr_lig= int(img.shape[0])     # TROUVEZ LES DIMENSIONS D'IMAGE
        nbr_col= int(img.shape[1])
        local_lig= int(no_rows/size)   # DÉFINIR LES DIMENSIONS LOCALES
         local_col= int(no_cols/size)
        local_x = np.empty((no_rows, local_c),dtype='uint8')
         # DÉFINIR UNE IMAGE LOCALE À DIFFUSER SUR CHAQUE NOEUD
        Inarray = np.array([no_rows, local_c])
         # DÉFINIR UNE IMAGE LOCALE À DIFFUSER SUR CHAQUE NOEUD
else:
        nv_img  = None  # DÉFINIR LES VALEURS PAR DÉFAUT POUR LES VARIABLES
        nbr_lig= None
        img     = None
        local_col= None
        Inarray = None
```

- **ETAPE 3, 4:**

Le nœud maître est maintenant configuré. Maintenant, nous devons exécuter la commande bcast dans chaque nœud pour diffuser les dimensions de l'image locale, puis la commande Scatterv pour disperser différents morceaux d'images sur différents nœuds. Notez que nous utilisons Scatterv ici pour autoriser des dimensions d'image irrégulières lors de la diffusion. Nous calculons nos dimensions d'image locale comme local_r = int (no_rows / size) et local_c = int (no_cols / size) qui peuvent ne pas toujours diviser l'image également. Donc, Scatterv s'assure que nous diffusons l'image entière quelle que soit la taille. Nous définirons un morceau d'image local comme local_x.

```
## FAITES LE RESTE DU CODE DANS CHAQUE NOEUD ##
t4 = MPI.Wtime()
#DÉMARREZ LA MINUTERIE POUR TROUVER LE TEMPS REQUIS POUR EXÉCUTER LE CODE
Inarray = comm.bcast(Inarray, root = 0)
# DIFFUSION DES DIMENSIONS D'IMAGE LOCALES SUR CHAQUE NOEUD
nbr_lig= Inarray[0]
local_col= Inarray[1]
local_x = np.empty((nbr_lig, local_col),dtype='uint8')
# DÉFINIR UNE IMAGE LOCALE SUR CHAQUE NOEUD
comm.Scatterv(img,local_x,root = 0)    # SCATTER L'IMAGE A CHAQUE NOEUD
```

Maintenant que l'image est dans les nœuds respectifs, nous pouvons compter le nombre d'étoiles dans chacun d'eux. Nous allons d'abord définir deux variables local_star_count et total_star_count pour contenir le nombre d'étoiles local et total.

```
nbr_local_etoiles = np.arange(1)
# DÉFINIR LES VARIABLES LOCALES ET TOTALES DU COMPTE D'ÉTOILES
nbr_total_etoiles count = np.arange(1)
```

- **ETAPE 5:**

Une etoile dans l'image est trouvè par l'utilisation de la methode adaptivethreshold dans opencv.les lignes de code suivantes font de meme.

```
img_thresh =
cv2.adaptiveThreshold(local_x,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,5
9,0) # LOGIQUE POUR COMPTER LES ÉTOILES
nbr_local_etoiles = ((200 < img_ _thresh)).sum()
#LOGIQUE POUR COMPTER LES ÉTOILES DANS CHAQUE NOEUD
```

- **ETAPE 6:**

Maintenant nous avons le nombre d'etoiles dans chacun des noeuds.nous allons les afficher avec le rang du noeud.ensuite,le nombre d'etoiles est calculè à l'aide de la function reduce et à l'aide de l'operation MPI.SUM sur le noeud maître.

```
print ("le nombre des etoiles dans processeur ", rang, "est: ",
nbr_local_etoiles)
# AFFICHER LA VALEUR DU NOMBRE D'ÉTOILES DANS CHAQUE NOEUD
comm.Reduce(nbr_local_etoiles, nbr_total_etoiles, op = MPI.SUM, root = 0)
# REDUCE LE COMPTE D'ÉTOILES LOCAUX EN VARIABLE DE COMPTE D'ÉTOILES TOTAL À L'AIDE
DE LA FONCTION REDUCE
```

- **ETAPE 7:**

Enfin ,dans le noeud maître,le temporisateur est arretè,le temps est enregistrè et affichè avec le nombre totale d'etoiles,l'image est affichèe,si necessaire.

```
if rang == 0:                    # DANS LE NŒUD MAÎTRE
        t5 = MPI.Wtime()         # arrêter le chronomètre
        print("temps d'execution du programme est: ", t5-t4 )
         # AFFICHER LE TEMPS NÉCESSAIRE POUR EXÉCUTER LE CODE
        print("le nombre total des etoiles: ", nbr_total_etoiles)
         # AFFICHER LE NOMBRE TOTAL D'ÉTOILES
```

4. EXECUTION DU CODE SEQUENTIEL VS PARALLELE:

4.1 EXECUTION DU CODE SEQUENTIEL:

```
temps d'execution du programme est: 2095.690460205078 secondes
le nombre total des etoiles est :  190369843
```

4.2 EXECUTION DU CODE PARALLELE:

COMMANDE: mpiexec -n 10 python MPI_TP.py (pour 10 noeuds/Processeurs)

```
temps d'execution du programme est:  108.92766599999959
le nombre total des etoiles:  [190369843]
```

5. CONCLUSION:

Ce TP se concentre sur la puissance de MPI et d'OpenCV pour accélérer considérablement le temps de traitement requis lors de la manipulation des images.