OS202

Examen Machine

Taulois Braga Bernardo

Mars 2024

Table des matières

1 Configuration de l'ordinateur **1**

2 Paralléliser le code **1**

1 Configuration de l'ordinateur

— Nombre de coeurs de calcul : 4 ;

— Mémoire cache :

  — L1d : 192 KiB (4 instances) ;

  — L1i : 128 KiB (4 instances) ;

  — L2 : 5 MiB (4 instances) ;

  — L3 : 12 MiB (1 instance).

2 Paralléliser le code

Pour partitionner l'image en *nbp* processus, les images sont chargées dans chacun des processus et ensuite séparées en lignes. Un exemple est l'utilisation de l'image en gris, séparée entre les processus avec le code :

```
# NOMBRE DE LIGNES POUR CHAQUE PROCESSUS
local_rows_values_gray = values_gray.shape[0] // nbp
# SEPARER L'IMAGE ENTRE LES PROCESSUS
local_values_gray = (values_gray[(local_rows_values_gray * rank
   ):(local_rows_values_gray * (rank + 1))][:][:])
```

Cela est fait tout au long du code et donne comme résultat les images dans la Figure 1. On voit que le résultat est cohérent avec ce qui a été proposé.

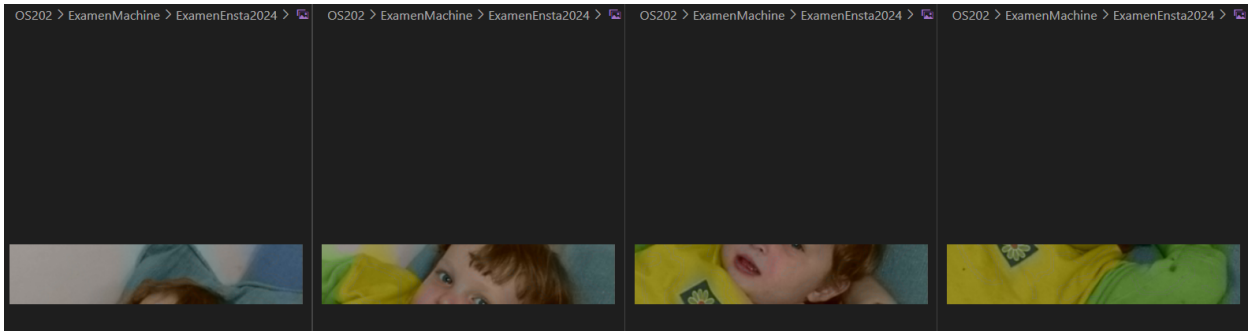

FIGURE 1 – Séparation de la figure en 4 processus.

On pourrait même, dans ce cas, faire une communication à la fin des calculs pour avois une image complète dans chaque processus. Cela pourrait être fait avec la fonction `ALlgather` de MPI.

```
new_image_array_global = np.empty((shape[0] * nbp,shape[1],3),
   dtype=np.uint8)
globCom.Allgather(new_image_array, new_image_array_global)
new_im = Image.fromarray(new_image_array_global, mode='YCbCr')
```

En revanche, on peut noter dans l'image finale quelques courbes de points gris. Cela correspond aux limites de les marques faites dans `example_marked.bmp`. On peut en déduire que, comme les limites de ces dessins sont à la frontière de deux régions très différentes, ils auront un gradient très élevés et seront colorés avec gris.

Ensuite, on a parallélisé les opérations de produit matrice-vecteur et l'algorithme de gradient conjugué. Ces opérations sont faites dans la fonction `minimize`. On a choisit de paralléliser les opérations de matrices en ligne :

```
new_line = x0.shape[0] // nbp
A_local = A[(new_line * rank):(new_line * (rank + 1))][:]
local_product = A_local.dot(x0)
A_dot_x0 = np.empty(x0.shape[0])
globCom.Gather(local_product, A_dot_x0, root=0)
globCom.Bcast(A_dot_x0, root=0)
```

Ainsi, tous les processus doivent avoir l'accès aux résultats avec les fonctions `Gather` et `Bcast`. L'image n'est composée que dans le processus 0. Le résultat est montré dans la Figure 3, et est cohérent avec l'attendu.

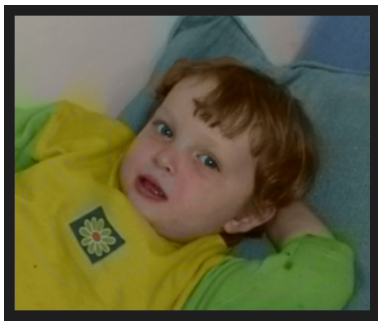

FIGURE 2 – Résultat de parallélisation.

On peut comparér les speedups ci-dessus et on observe que le meilleur résultat se trouve quand on sépare la matrice en blocs, vu que le problème et plus memory-bound que CPU-bound.

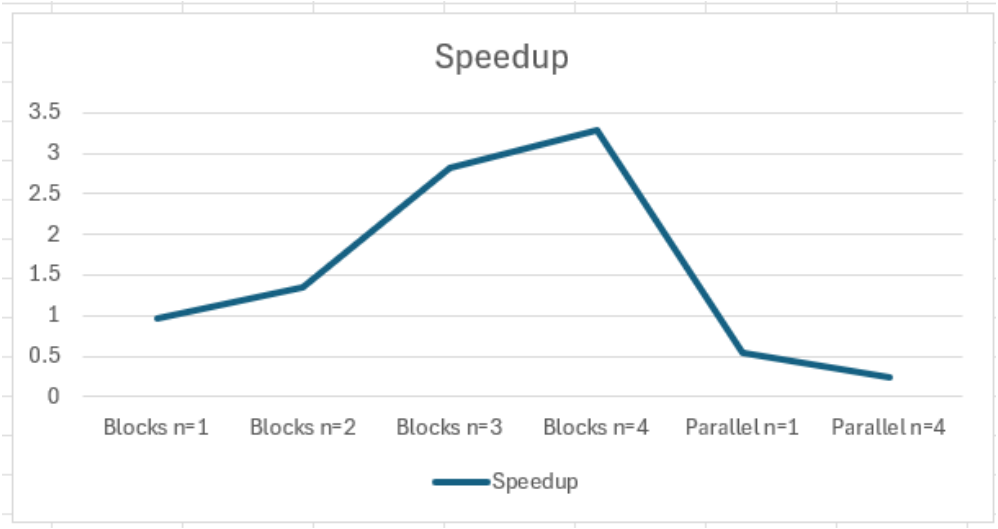

FIGURE 3 – Comparaison de speedup.