**NOM Prénom étudiant 1 :**

**NOM Prénom étudiant 2 :**

**Outils pour la synchronisation**

**TP Noté**

L’objectif de ce TP est de résoudre un problème pouvant tirer parti de la parallélisation. Vous devrez rendre pour le **19/03/2019 :**

* vos codes sources en respectant le nom des fichiers c et les arguments des programmes dont l’ordre et le nombre sont imposés,
* ce rapport complété (le symbole  indique qu’une réponse est attendue).

**Exercice 1 : Traitement d’une image**

Dans ce TP nous allons utiliser une librairie disponible sur internet (*qdbmp.h)* afin de manipuler des images au format bmp pixel par pixel.

**Question 1.1 :** Le fichier image.c vous montre comment lire une image bmp, la passer en noir et blanc et l’enregistrer. Compilez, linkez et exécutez le programme image.c en utilisant les lignes de commande suivantes :

**gcc -c qdbmp.c**

**gcc -c image.c**

**gcc qdbmp.o image.o -lpthread -o im**

**./im "pepper.bmp" "pepper\_nb.bmp"**

**Question 1.2 :** Proposez une version parallèle du programme image.c avec threads, de façon à ce que chaque thread traite lignes de l’image ( = hauteur de l’image). Nommez votre programme exercice1.c. Le nombre sera récupéré en argument de votre programme. On pourra exécuter votre programme avec la commande suivante :

**./exercice1 "pepper.bmp" "pepper\_nb.bmp" 4**

**Question 1.3 :** Quel est le type de stratégie de décomposition mis en place ?

**Question 1.4 :** Donnez le graphe de dépendance des tâches et le graphe d’interaction pour ce programme.

**Exercice 2 : Threads maître/esclaves**

Dans le programme précèdent, c’est le thread principal (main) qui se charge d’ouvrir et enregistrer le fichier bmp.

**Question 2.1 :** Proposez une nouvelle version du programme dans laquelle un thread dit « maître » se charge de lire le fichier bmp, lance les threads qui convertissent l’image en noir et blanc, puis attend la terminaison des threads avant d’enregistrer l’image. Nommez votre programme exercice2.c. On pourra exécuter votre programme avec la commande suivante :

**./exercice2 "pepper.bmp" "pepper\_nb.bmp" 4**

**Question 2.2** : Quel est l’intérêt de cette approche ?

**Question 2.3 :** Donnez le graphe de dépendance des tâches et le graphe d’interaction pour ce programme.

**Exercice 3 : Traiter un dossier**

Le programme précèdent traite une image à la fois.

**Question 3.1** : Créez un dossier que vous appelez *data*. Copiez dans ce dossier l’image pepper.bmp. Exécutez la commande suivante pour dupliquer 30 fois l’image pepper.bmp :

**for (( i=1 ; i<=30 ; i++ )); do cp pepper.bmp pepper$i.bmp; done**

**Question 3.2** : Modifiez le programme précèdent de façon à traiter toutes les images contenues dans le dossier *data*. Utilisez la librairie *dirent.h* pour cela. Nommez votre programme exercice3.c. On pourra exécuter votre programme avec la commande suivante :

**./exercice3 "data" 4**

**Question 3.3 :** Mesurer le temps d’exécution de votre programme pour différentes valeurs de et donnez les résultats sous forme d’un graphique. Commentez.

**Question 3.4 :** Créer une version séquentielle de votre programme et appelez-le exercice3\_sequentiel.c. On pourra exécuter votre programme avec la commande suivante :

**./exercice3\_sequentiel "data"**

**Question 3.5 :** Mesurez l’accélération de la version parallèle de exercice3.c pour différentes valeurs de et donnez les résultats sous forme d’un graphique. Commentez.

**Exercice 4 : Limiter le nombre de threads maîtres**

Dans cet exercice on ne veut pas qu’il y ait plus de threads maîtres lancés à un moment donné.

**Question 4.1** : Modifiez le programme précèdent de façon à prendre en compte cette contrainte. Nommez votre programme exercice4.c. On pourra exécuter votre programme avec la commande suivante :

**./exercice4 "data" 4 2**

**Question 4.2 :** Expliquez comment vous avez fait pour prendre en compte cette contrainte.

**Conclusion :** Que retenez-vous de ce TP ? Qu’avez-vous appris ?