UNIVERSITÉ D'ORLÉANS **M1 Info**

Informatique Année **2019 - 2020**

Projet **QuickSort**

L'objectif de ce projet est de paralléliser l'algorithme de tri appelé **QuickSort** en suivant les algorithmes vus en cours et décrits dans le chapitre du livre "Introduction to Parallel Computing" de A. Grama, A. Gupta, G. Karypis et V.Kumar. Vous pourrez trouver ici des éléments sur cet algorithme et sinon le chapitre est également disponible sur Celene.

Vous devez travailler **en binôme** et les différentes étapes sont décrites ci-dessous ainsi que les rendus, des archives au format .tgz, à faire au cours de la journée. Pour vous aider une version séquentielle de l'algorithme est disponible sur Celene. Attention ne prenez pas cette version pour en faire une version OpenMP mais suivez le déroulement ci-dessous.

1 Partie 1 : Version QuickSort en OpenMP

1.1 Les calculs élémentaires

Dans un premier temps vous devez écrire les fonctions réalisant les opérations élémentaires d'une étape de l'algorithme QuickSort.

1. La fonction `Partitionnement` qui permet de répartir les éléments à droite ou à gauche du pivot. Chaque thread devra effectuer le partitionnement d'un morceau du tableau global. La signature à respecter est la suivante

   ```
   int Partitionnement(int* tab, int n, int pivot, int nb_threads, int* s, int* r);
   ```

   - tab est le tableau de données
   - n sa longueur
   - pivot est le pivot que vous choisirez avec la méthode de votre choix
   - nb_threads est le nombre de threads souhaité
   - s est un tableau tel que s[i] est le nombre d'éléments plus petits ou égaux au pivot pour le ième morceau du tableau $0 \leq i < nb\_threads$
   - r est un tableau tel que r[i] est le nombre d'éléments plus grand que le pivot pour le ième morceau du tableau $0 \leq i < nb\_threads$

2. La fonction `SommePrefixe` qui permet de déterminer les tableaux somme_left et somme_right de taille $nb\_threads + 1$ tels que somme_left[i] (respectivement somme_right[i]) est le nombre d'éléments plus petits ou égaux (respectivement plus grands) que le pivot et qui ont été trouvés par les threads qui précédent le thread i (somme_left[0] = somme_right[0]=0). La signature à respecter est la suivante

   ```
   void SommePrefixe(int* s, int* r,
   int* somme_left, int* somme_right, int nb_threads);
   ```

3. La fonction `Rearrangement` qui permet de reconstruire correctement un nouveau tableau tel que tous les éléments plus petits ou égaux au pivot soient à gauche dans le tableau et les autres à droite. La signature à respecter est la suivante

   ```
   void Rearrangement(int* somme_left, int* somme_right, int* tab, int* res,
   int n, int nb_threads);
   ```

   - res est le tableau résultat du réarrangement.
   - les autres paramètres sont similaires aux fonctions précédentes.

Premier rendu à 10h30
Ce premier rendu doit contenir les fonctions ci-dessus ainsi qu'un programme principal permettant de vérifier leur bon fonctionnement. Vous devez également fournir le Makefile associé et un README indiquant la constitution du binôme, un bilan sur le travail réalisé et la manière d'exécuter votre programme.
Correction proposée à 10h45 Une correction est disponible sur Celene à partir de 10h45. Vous n'êtes pas obligés de la prendre en compte. Il est préférable d'utiliser ses propres développements mais si nécessaire vous pouvez aussi vous approprier les fonctions proposées.

1.2 Mise en place de l'algorithme récursif

Cette partie consiste à finaliser votre développement et à mettre en place l'algorithme récursif qui permet de réaliser le tri. Vous êtes libres de réorganiser votre code comme vous le souhaitez.

Deuxième rendu à 13h Pour ce deuxième rendu, l'archive doit contenir l'ensemble des implémentations avec un Makefile et un programme principal permettant de vérifier que tout fonctionne. Vous devez également fournir un nouveau README donnant un bilan sur le travail réalisé et la manière d'exécuter votre programme.

2 Partie 2 : Version QuickSort en MPI

Cette dernière partie consiste à développer le tri QuickSort en MPI. L'idée est également de découper ce travail en deux parties avec l'implémentation des opérations élémentaires et ensuite la mise en place de l'algorithme récursif complet.

Troisième rendu à 16h La troisième archive à rendre doit contenir les fonctions des opérations élémentaires à effectuer. Vous êtes libres de définir leur signature mais vous devez rajouter un programme principal permettant de vérifier leur bon fonctionnement. Vous devez également fournir le Makefile et un README indiquant la constitution du binôme, une explication des fonctions développées (signature et rôle) et comment exécuter votre code.

Troisième rendu à 18h La dernière archive doit contenir l'implémentation complète du QuickSort en MPI. Avec toujours un README bilan et les éléments permettant de faire tourner votre code et de vérifier que tout fonctionne bien.