# MS-SIS - Programmation parallèle

(2018-2019)

# TP1

OpenMP

julien.jaeger@cea.fr

Les objectifs de ce TP sont :
— Compréhension du modèle déxécution
— partage de travail (parcours de tableau)
— parallélisation d'un code casseur de mdp

## I Modèle d'exécution

Dans cette partie nous considérons le fichier *$PATH/CODE/Ex1/print_rank_1.c*. À tout moment, vous pouvez compiler le fichier et l'exécuter (gcc print_rank_1.c -o print_rank_1.pgr -fopenmp && OMP_NUM_THREADS=8 ./print_rank_1.pgr).

**Q.1:** Ce programme affiche l'identifiant unique (rank) du thread courant parmis tous les threads demandés. Que font les fonctions `omp_get_num_threads()` et `omp_get_thread_num()` ?

**Q.2:** Pour le moment, les appels à ces fonctions et le print ne sont pas dans une région parallèle. Insérer une région parallèle OpenMP avec le pragma `#pragma omp parallel`.

**Q.3:** Insérer une barrière OpenMP (`#pragma omp barrier` avant le print. Que se passe-t-il ? Pourquoi ? Comment peut-on corriger le problème ?

## II Partage de travail

Dans cette partie nous considérons le fichier *#PATH/CODE/Ex2/parallel_for_1_manual.c*. À tout moment, vous pouvez compiler le fichier et l'exécuter (gcc parallel_for_1_manual.c -o parallel_for_1_manual.pgr -fopenmp && OMP_NUM_THREADS=6 ./parallel_for_1_manual.pgr).

**Q.4:** Le programme parcourt un tableau d'entier et réalise la somme de chaque entier. Exécuter-le. Quel est le problème ?

**Q.5:** Répartir le travail de parcourt de tableau et de somme entre les threads en utilisant le rang du thread. Utiliser une variable temporaire pour stocker la somme partielle, avant de sommer les sommes partielles pour obtenir la valeur finale. Penser à protéger la somme finale (avec un atomic ou une section critique).

**Q.6:** Il est possible de répartir automatiquement les itérations d'une boucle entre les différents threads avec le pragma `#pragma omp for`. Utiliser ce pragma pour remplacer votre découpage manuel.

**Q.7:** Au lieu de protéger la somme finale avec une section critique, il est possible de spécifier à une région parallèle (ou une boucle for) qu'une réduction à lieu dans celle-ci. Utiliser cette fonctionnalité.

**Q.8:** Il est possible de fusionner les pragmas `#pragma omp parallel` et `#pragma omp for` en un seul pragma. Supprimer la boucle permettant l'affichage des sommes partielles, et utiliser ce pragma combiné.

# III Paralléliser un code casseur de mdp

Dans cette partie nous considérons le fichier i*$PATH/CORRECTION/Ex3/breaker_for.c*. À tout moment, vous pouvez compiler le fichier et l'exécuter (gcc breaker_for.c -o breaker_for.pgr -fopenmp -lm -lcrypt && ./breaker_for.pgr).

**Q.9:** Le programme fait une recherche gloutonne pour trouver un mot de passe à partir du mot de passe crypté. Pour le moment, le code test toutes les possibilités avec une boucle for. Paralléliser cette boucle pour que la recherche soit plus rapide. Ouvrir la page de man de la fonction `crypt` pour vérifier si celle-ci peut être utilisé en parallèle.

# IV DEVOIR MAISON

Cette partie est à réaliser pour le prochain cours.

**Q.10:** Ce programme est le même que précédemment, à l'exception qu'il s'arrête dés que le mot de passe est trouvé. Paralléliser ce code avec la même optimisation : dés qu'un thread trouve le mot de passe, il arrête le travail de tous les threads.