# TP Architecture Matérielle - Parallélisme

## Objectifs

Nous allons explorer l'impact conjoint du parallélisme et de la gestion saine des caches sur un exemple simple. Le code fournit (matrix.hpp et inner.cpp) contient le code pour gérer des matrices 2D et une fonction (inner_sum) permettant de calculer la somme de chaque ligne d'une matrice et de renvoyer la matrice des résultats.

Nous utiliserons OpenMP pour la phase de parallélisation. OpenMP est accessible via l'en-tête `<omp.h>`. La parallélisation d'une boucle simple se fait en ajoutant la clause :

```
#pragma omp parallel for
```

Juste immédiatement avant la boucle for que l'on désire paralléliser. Un programme OpenMP se compile et se link avec l'option -fopenmp :

```
g++ monfichier.cpp -o monexe -fopenmp -O3
```

## Exercice 1

Compiler le fichier inner.cpp et exécutez le. Le programme obtenu prend en paramètre de ligne de commande les deux dimensions de la matrice et le nombre de répétitions de la mesure de performances. Mesurez le temps d'exécution pour des tailles variants de très petits (4x4) à très grand (1024x1024) en n'oubliant pas de tester les cas non-carrés.

Qu'observez-vous ? Déterminez la fréquence de votre processeur et déduisez-en une mesure en nombre de cycles CPU par valeurs calculées. Vérifiez que le comportement de votre fonction est cohérent avec la taille de votre cache L2.

## Exercice 2

Implémentez dans le fichier outer.cpp une fonction qui calcule la somme des valeurs de la matrice par colonnes en utilisant un algorithme naïf parcourant chaque colonne, puis chaque ligne. Effectuez ensuite les mêmes mesures de performances que dans l'exercice précédent.

Vérifiez que le comportement de la fonction est cohérent avec le cas d'une fonction ne respectant pas l'ordre correct du parcours des caches

# Exercice 3

Implémentez dans le fichier good_outer.cpp une fonction good_outer_sum qui calcule la somme des valeurs de la matrice par colonne en utilisant un algorithme plus efficace parcourant chaque ligne, puis chaque colonne. Effectuez ensuite les mêmes mesures de performance que dans l'exercice précédent.

Vérifiez que le comportement de la fonction est cohérent avec le cas d'une fonction respectant l'ordre correct du parcours des caches. Comment expliquez-vous la différence de performance entre cette version et la version de l'exercice 1 ? Vérifiez votre hypothèse en générant le code assembleur des deux programmes et en examinant le contenu de la boucle interne. L'accès au code assembleur peut se faire soit en copiant/collant le code sur gcc.godbolt.org soit en utilisant l'option -S de g++.

# Exercice 4

Pour chacune des trois versions de la somme partielle de la matrice implémentez ci-dessus, utilisez OpenMP pour paralléliser le code de calcul.

Quelle accélération et quelle efficacité obtenez-vous, sachant que par défaut, OpenMP utilise tous les threads disponibles sur votre machine ? Modifiez la variable d'environnement OMP_NUM_THREADS avant l'exécution de votre programme pour vérifier son comportement.

```
OMP_NUM_THREADS=4 ./mon_executable 15 35 1000
```

Que se passe-t-il en terme de performances si vous appliquez la directive OpenMP sur la boucle for la plus interne au lieu de la boucle la plus externe ? Que se passe-t-il si vous utilisez une directive OpenMP sur les deux ?

Déterminez la taille minimale de la matrice en dessous de laquelle vous n'obtenez pas d'accélération. Pour quelle raison une telle taille minimale existe-t-elle ?