# L3 MI / PROG
# TP#0

2022-W36

## Exercice 1 : produit scalaire, compilation

Ce court exercice a pour but de constater l'impact des optimisations de compilation sur un programme très simple.

En fonction de votre maitrise de la compilation séparée, vous pouvez n'écrire vos fonctions que dans un seul fichier ou bien adopter la structure indiquée dans les questions. Dans ce dernier cas, vous pouvez également écrire un `Makefile` pour piloter votre processus de compilation.

**Rappels :**

- Pour utiliser les types `uint32_t`, `uint64_t`, il faut inclure le fichier `stdint.h` via `#include <stdint.h>`.
- Pour afficher les mêmes types via `printf` sur une machine 64 bits, utilisez les formats d'affichage `%u` et `%lu` ou `%llu` respectivement.
- Pour mesure le temps d'exécution d'un programme `prog`, dans un shell UNIX, faire `time prog`.

**Q.1 :** Implémentez les fonctions de calcul de produit scalaire suivantes pour des vecteurs de dimension `dim` et à coefficients de types respectivement `double`, `float`, `uint64_t`, `uint32_t` :

- `double psd(double x[], double y[], int dim);`
- `float psf(float x[], float y[], int dim);`
- `uint64_t psu64(uint64_t x[], uint64_t y[], int dim);`
- `uint32_t psu32(uint32_t x[], uint32_t y[], int dim);`

dans un fichier `ps.c`, et déclarez les dans un fichier `ps.h` correspondant.

**Q.2 :** Implémentez les fonctions de test suivantes :

- `void test_double(int dim, int repet)`
- `void test_float(int dim, int repet)`
- `void test_u64(int dim, int repet)`
- `void test_u32(int dim, int repet)`

qui déclarent deux vecteurs `x` et `y` de type approprié et de taille `dim` (par ex. via la construction `type x[dim], y[dim];`), les initialisent avec tous leurs coefficients à 1, calculent `repet` fois leur produit scalaire en dimension `dim`, accumulent (c-à-d somment) le résultat (dans une variable du type approprié, initialisée à zéro), et l'affichent sur la sortie standard.

Ces fonctions devront être écrites dans un fichier `psm.c`, qui comportera aussi votre fonction `main`.

**Q.3 :** Si vous êtes familier avec `make`, écrivez un `Makefile` vous permettant de compiler votre programme et de spécifier aisément le compilateur et les options de compilation à utiliser.

**Q.4 :** Lancez chacune de vos fonctions de tests avec les arguments 1000 et 10000000 avec au moins deux compilateurs (par ex. clang et gcc) et les options suivantes :

- `-O0`
- `-O2`
- `-O2 -march=native`

Comment pouvez-vous expliquer : 1) les temps de calcul obtenus ; 2) les différentes valeurs des résultats ?