TP 8 : Optimisation de boucles et SIMD

Exercice 1 - Le retour du produit de matrices

Dans cet exercice, on considère que les matrices carrées sont représentées par le type `double*`.

Pour définir une matrice de taille $n \times n$, on alloue et on remplit un tableau de n^2 cases. Le coefficient à la ligne i et à la colonne j est alors le contenu de la case $(i-1) + n\,(j-1)$ du tableau.

1.1 Écrire un code `C` naïf pour effectuer le produit de deux matrices carrées.

1.2 Écrire un programme qui mesure le temps de calcul du produit de matrice naïf en fonction de la taille n récupérée à partir de la ligne de commande.

note : On utilisera la fonction `clock()` pour mesurer le temps de calcul.

1.3 Comment échanger les boucles pour minimiser le temps de calcul de votre code ?

Justifier le choix de l'échange, puis valider le expérimentalement en mesurant le temps de calcul d'une nouvelle implantation du produit de matrices.

Exercice 2 - Somme de vecteurs

L'objectif de cet exercice est d'améliorer le calcul de la somme de deux vecteurs d'entiers codés sur 8 bits (type `uint8_t`).

Pour pouvoir utiliser les jeux d'instructions SSE2 et AVX2, on compilera respectivement avec les options `-msse2` et `-mavx2`.

Pour rappel, une documentation des fonctions intrinsèques déclarées dans `x86intrin.h` est disponible sur cette page :

`https://software.intel.com/sites/landingpage/IntrinsicsGuide/`

2.1 Écrire une fonction `C` qui prend en argument 3 tableaux C, A, et B, et leur taille commune n, et qui écrit dans C la somme (case par case) de A et B.

2.2 On suppose à partir d'ici que n est un multiple de 16.

Utiliser le type `__m128i` et le jeu d'instructions SSE2 pour accélérer les calculs.

2.3 On suppose à partir d'ici que n est un multiple de 32.

Utiliser le type `__m256i` et les jeux d'instructions AVX et AVX2 pour accélérer les calculs.

2.4 Comparer les temps de calculs des différentes versions. Commenter.

Exercice 3 - Tableaux et structures

Dans cet exercice, nous allons travailler avec des points (couples de `floats`) dont les coordonnées sont comprises entre 0 et 1. Il s'agit pour chaque question d'organiser les données de façon adéquate afin de pouvoir profiter au maximum du jeu d'instructions SSE.

3.1 Écrire un programme calculant le barycentre d'une séquence de n points.

3.2 Écrire un programme calculant la plus petite distance (en norme 2) entre un point donné en argument et un point parmi une séquence de n points donnée en argument.