TP HPC 1

Master Informatique - 1ère année

Année 2020-2021

L'objectif de ce TP est de tester l'intérêt des routines AVX sur le cas simple du calcul d'un produit scalaire. On rappelle que le produit scalaire de deux vecteurs u et v de dimension n se calcule de la manière suivante :

$$s = \sum_{i=1}^{n} u_i \times v_i$$

Exercice 1

Récupérez la version initiale du fichier source (`ps.cpp`) joint à cet énoncé. Vous aurez à le compléter progressivement au cours de ce TP.

Ce fichier contient une fonction nommée `ps_float(int size)` qui, à ce stade, effectue le produit scalaire séquentiem entre deux vecteurs réels de taille `size`. Ces vecteurs sont créés et intialisés dans la fonction et le produit scalaire est effectué plusieurs fois via une boucle, afin de pouvoir avoir des temps de calcul significatifs. Les paramètres permettant de jouer sur le temps de calcul, définis comme des constantes dans le fichier source, sont alors :

- `NB` le nombre de fois que le calcul du produit scalaire sera évalué ;
- `SIZE` la taille des vecteurs à utiliser. Sa valeur devra nécessairement être un multiple de 8.

Travail à réaliser : mettez en place dans la fonction `ps_float()` le code permettant de mesurer le temps de calcul des `NB`évaluations du produit scalaire. Le temps obtenu sera affiché à l'écran, sous la forme :

`temps ps float seq = XXX secondes - ps = YYY`

où `XXX` sera remplacé par le temps de clacul obtenu et `YYY` par la valeur du produit scalaire calculé. Vous ferez différents tests sur votre machine pour régler les paramètres `NB` et `SIZE`[1] de manière à ce que le temps de calcul n'excède pas quelques secondes.

Exercice 2

Dans cet exercice, vous allez être amenés à compléter la fonction `ps_float` de telle sorte qu'elle effectue le calcul du produit scalaire à l'aide des routines AVX. Voous allez procéder en suivant les différentes étapes ci-après :

1. Comme pour la version séquentielle, le produit scalaire vectoriel sera calcule `NB` fois. A chaque tour de boucle, la valeur du produit scalaire (la somme du produit des composantes) sera remis à 0. Ici, il faudra cependant utiliser une variable de type `__m256` pour stocker le résultats des 8 sommes qui seront faites en parallèles.

   Application : Créez la variable qui contiendra les sommes, ajoutez la boucle des `NB` évaluations et initialisez la variable à chaque entrée dans la boucle. Vous utiliserez l'*intrinsic* `_mm256_setzero_ps` en vous référant à la documentation ;

2. Les types vectoriels comme `__m256` contiennent plusieurs valeurs accolées dans la largeur de la donnée. Il peut être utile, lors d'opérations de mise au point, d'afficher ces valeurs. Il faut alors écrire une fonction spécifique, le C++ ne pouvant les prendre en charge directement avec l'opérateur `<<`.

1. Il est conseillé de ne pas utiliser une taille trop grande des vecteurs, qui provoquerait alors des défauts de cache et une baisse notable des performances

Application : Complétez le code la fonction `printf(std::string s, __m256 *v)` prévue dans le fichier source, de telle sorte qu'elle affiche les 8 valeurs réelles contenues dans son second paramètre. Le paramétre `s` représentera le message qui sera affiché devant les 8 valeurs, pour indiquer leur signification. Vous modifierez ensuite votre code manière à ce qu'il affiche le contenu de la somme uniquement au premier tour de boucle, après son initialisation (normalement, 8 zéros ...). Testez avant de poursuivre ...

3. Complétez à présent votre fonction afin qu'elle calcule effectivement la somme vectorielle des composantes des 2 vecteurs. Les *instrinsics* à utiliser sont `_mm256_mul_ps` et `_mm256_add_ps` ;
4. A la sortie du calcul, vous disposez de la somme vectorielle du produit des différentes composantes ; Il faut à présent sommer les valeurs contenues dans cette somme vectorielle pour obtenir la valeur du produit scalaire final. Compléter votre code en ce sens et faites afficher la valeur obtenue pour vérifier qu'elle est bien identique à celle calculée avec le produit scalaire séquentiel ;
5. Enfin, complétez votre code de manière à pouvoir évaluer le temps de calcul de la partie vectorielle de la fonction, l'afficher avec le même format que pour la partie séquentielle et enfin afficher l'accélération obtenue.

Exercice 3

Deéveloppez les fonctions `printd(std::string s, __m256d *v)` et `ps_double(int size)` sur le modèle des fonctions similaires de l'exercice précédent.

Exercice 4

Faire de même pour des vecteurs d'entiers.