SIMD TP1
Introduction aux intrinsics

La liste des intrinsics SSE/AVX est disponible à l'adresse suivante : `https://software.intel.com/sites/landingpage/IntrinsicsGuide/`

Pour déterminer le type d'unité SIMD présente sur votre processeur taper :

```
cat /proc/cpuinfo | grep flags | uniq
```

Sur un processeur Broadwell i5-5300U la sortie est la suivante :

```
flags : fpu ... sse sse2 ... pni ... ssse3 ... fma ... sse4_1 sse4_2
... avx... avx2 ...
```

Tous les jeux d'instructions SSE sont disponibles ainsi qu'AVX, AVX2 et FMA.

Pour indiquer au compilateur (GCC ou Clang) quel jeu d'instruction est utilisé il faut ajouter un flag, par exemple `-msse4.1` ou `-mavx`. Par défaut les compilateurs récents activent le jeu d'instruction SSE2.

1 Exercices simples

1.1 Copie

Voici l'exemple de base effectuant la copie de 4 floats d'un tableau dans un autre tableau :

```
1  /**
2   * Copy values from an array to another.
3   */
4  #include <immintrin.h>
5
6  #include <iostream>
7
8
9  int main()
10 {
11    // Static arrays are stored into the stack thus we need to add an alignment attribute to tell the compiler to correctly
   align both arrays.
12    float array0[ 4 ] __attribute__((aligned(16))) = { 0.0f, 1.0f, 2.0f, 3.0f };
13    float array1[ 4 ] __attribute__((aligned(16)));
14
15    // Load 4 values from the first array into a SSE register.
16    __m128 r0 = _mm_load_ps( array0 );
17    // Store the content of the register into the second array.
18    _mm_store_ps( array1, r0 );
19
20    for( auto x: array1 )
21    {
22      std::cout << x << std::endl;
23    }
24
25    return 0;
26 }
```

Pour compiler cet exemple utiliser la commande `g++ -o copy-sse copy-sse.cpp`.

Généraliser cet exemple à un tableau de taille arbitraire, les tableaux seront alloués dans le tas et alignés à l'aide de la fonction `_mm_malloc`.

1.2 Max

Trouver la valeur maximale d'un tableau en reprenant l'exemple précédent et en utilisant l'intrinsic `_mm_max_ps`.

1.3 Reduce

Calculer la somme des éléments d'un tableau.

1.4 Produit scalaire

Calculer le produit scalaire de 2 vecteurs de doubles.

2 Manipulation des entiers

2.1 Addition

Pour les unités vectorielles SSE/AVX, il y a un seul type de registre (`__m128i __m256i`) pour tous les types entiers. Ce sont donc les intrinsics utilisés qui définissent le type d'entiers manipulé (suivant leur préfixe). Additionner 2 vecteurs d'entiers 16 bits non signés (`unsigned short`).

2.2 MAJUSCULE

Transformer les caractères minuscules d'une chaîne (`std::string`) en majuscule. Pour cela il suffit de mettre le 5ème bit d'un caractère minuscule à 0 (cf. table ASCII). En version vectorielle, on ne peut pas utiliser de `if` pour tester les caractères un par un, on utilise donc des masques et des opérations logiques. Voici l'algorithme scalaire à adapter :

```
Pour chaque caractère c:
  mask0 = ( c < 'a' || c > 'z' ) - 1 // contient uniquement des 0 si c est
                                     // minuscule ou uniquement des 1 sinon.
  mask1 = 32 & !mask0 // négation de mask0 qui contient 32 aux positions minuscules,
                      // uniquement des 0 ailleurs.
  c = c & !mask1 // négation de mask1 qui contient la négation binaire de 32 aux
                 // positions minuscules, uniquement des 1 ailleurs.
  // Les caractères non minuscules ne sont donc pas affectés car 1 & x = x.
```

Les intrinsics à utiliser sont : `_mm_loadu_si128`, `_mm_storeu_si128`, `_mm_set1_epi8`, `_mm_cmplt_epi8`, `_mm_cmpgt_epi8`, `_mm_or_si128_mm_andnot_si128`