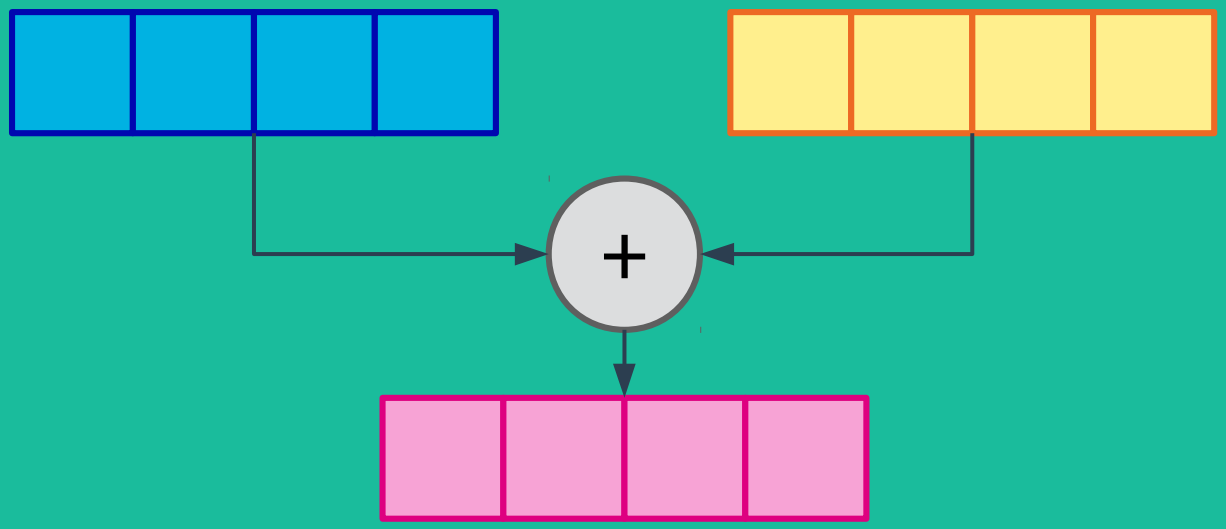

xsimd : SIMD explicite portable

Hadrien Grasland
CNRS – LAL

2018/06/21

Processus de vectorisation

- **Les instructions SIMD sont limitées et malpratiques**
  - Mieux vaut les réserver à son « fond de boucle »
  - Mieux vaut sous-traiter à une bibliothèque ce qu'on peut
- **Mais si on doit créer sa bibliothèque ?**
  - Interface centrée sur le domaine cible (algèbre, FFT…)
  - Algorithmes SIMD internes soigneusement optimisés
  - Couche inférieure qui abstrait les différences matérielles

Types d’abstractions matérielles

- **SIMD implicite : On ne manipule pas de vecteurs**
  - On exprime le parallélisme de données… et on prie
  - Exemples : OpenCL, ~~Intel ArBB~~, ~~Intel Cilk Plus~~…
- **SIMD explicite : On manipule des vecteurs abstraits**
  - Généralement via un type générique du style
    `VectorType<typename T, size_t N = MAX_HW_VEC<T>>`
  - Un peu moins d’élégance, beaucoup plus de contrôle
  - Exemples : VectorClass, UME::SIMD, xsimd…

Positionnement de xsimd

- **Un maillon de la pile QuantStack**
  - xsimd : Abstraction explicite du matériel SIMD
  - xtensor : Tableaux à N dimensions (« numpy en C++ »)
  - xtensor-blas : Binding vers les bibliothèques BLAS
- **Reprend les idées (et le code) de boost::simd**
  - Un projet open-source de NumScale (NT2, bSIMD…)
  - Récemment sabordé pour des raisons commerciales :-(

Opérations SIMD élémentaires

```cpp
#include <xsimd/xsimd.hpp>

namespace xs = xsimd;
using vec = xs::simd_type<float>;
// using sse_vec = xs::batch<float, 4>;

vec hello_world(const vec& x, const vec& y) {
  // xs::sin/cos respecte IEEE-754
  auto sin = xs::sin(x + y);
  auto cos = xs::cos(x - y);
  return xs::select(x < y, sin, cos);
}
```

Calcul complet : Vue d'ensemble

```cpp
#include <cstddef>
#include <vector>
#include "xsimd/xsimd.hpp"

namespace xs = xsimd;
using al_vec =
    std::vector<double,
                xsimd::aligned_allocator<double,
                                         XSIMD_DEFAULT_ALIGNMENT>>;

void mean(const al_vec& i1, const al_vec& i2, al_vec& o) {
  std::size_t size = i1.size();
  constexpr std::size_t simd_size = xs::simd_type<double>::size;
  std::size_t vec_size = size - size % simd_size;

  for(std::size_t i = 0; i < vec_size; i += simd_size) {
    auto bi1 = xs::load_aligned(&i1[i]);
    auto bi2 = xs::load_aligned(&i2[i]);
    auto bo = (bi1 + bi2) / 2;
    bo.store_aligned(&o[i]);
  }
  for(std::size_t i = vec_size; i < size; ++i) {
    o[i] = (i1[i] + i2[i]) / 2;
  }
}
```

Calcul complet : Gestion de l'alignement

```
1 #include <cstddef>
2 #include <vector>
3 #include "xsimd/xsimd.hpp"
4
5 namespace xs = xsimd;
6 using al_vec =
7     std::vector<double,
8                 xsimd::aligned_allocator<double,
9                                          XSIMD_DEFAULT_ALIGNMENT>>;
10
11 void mean(const al_vec& i1, const al_vec& i2, al_vec& o) {
12   std::size_t size = i1.size();
13   constexpr std::size_t simd_size = xs::simd_type<double>::size;
14   std::size_t vec_size = size - size % simd_size;
15
16   for(std::size_t i = 0; i < vec_size; i += simd_size) {
17     auto bi1 = xs::load_aligned(&i1[i]);
18     auto bi2 = xs::load_aligned(&i2[i]);
19     auto bo = (bi1 + bi2) / 2;
```

Calcul complet : Coeur du calcul

```
    std::vector<double,
               xsimd::aligned_allocator<double,
                                        XSIMD_DEFAULT_ALIGNMENT>>;

void mean(const al_vec& i1, const al_vec& i2, al_vec& o) {
  std::size_t size = i1.size();
  constexpr std::size_t simd_size = xs::simd_type<double>::size;
  std::size_t vec_size = size - size % simd_size;

  for(std::size_t i = 0; i < vec_size; i += simd_size) {
    auto bi1 = xs::load_aligned(&i1[i]);
    auto bi2 = xs::load_aligned(&i2[i]);
    auto bo = (bi1 + bi2) / 2;
    bo.store_aligned(&o[i]);
  }
  for(std::size_t i = vec_size; i < size; ++i) {
    o[i] = (i1[i] + i2[i]) / 2;
  }
}
```

Rappel : Si vous pouvez sous-traiter...

```
1 #include <xtensor/xtensor.hpp>
2
3 using data = xt::xtensor<float, 1>;
4
5 void mean(const data& i1, const data& i2, data& o) {
6   o = (i1 + i2) / 2;
7 }
```

Conclusion

- **xsimd fournit une abstraction SIMD explicite**
  - Souvent nécessaire pour des performances optimales
  - Supporte SSE/AVX/AVX512 et leux extensions, NEON…
- **Projet GSoC : De boost::simd à xtensor**
  - Départ : Algèbre basse dimensionnalité avec boost::simd
  - Portage vers xsimd effectué sans problème majeur
  - But : Optimiser xtensor jusqu'à pouvoir tout faire avec

Questions ?
Remarques ?