# TD SIMD

December 11, 2020

## Contents



Directives:

- Ce TD sera a rendre pour au plus tard Jeudi 17/12/2020 a 00:00 (minuit) aux adresses suivantes (selon qui est en charge de votre groupe):
  - *mohammed-salah.ibnamar@uvsq.fr*
  - *kevin.camus@uvsq.fr*
- Il faudra utiliser comme sujet du courriel le format suivant (a respecter a la lettre): `[ AoB TD_simd - NOM PRENOM ]`

Notes:

- Vous pouvez effectuer votre exploration sur une VM car l'execution du code n'est pas importante.

# 1 Introduction

L'objectif de ce TD est d'explorer le jeu d'instructions SIMD present sur les processeurs x86 (Intel et AMD) dont vous disposez.

L'architecture x86 propose des instructions vectorielles/SIMD sous la forme d'extensions:

- **SSE**: **Streaming SIMD Extensions** qui opere sur des vecteurs nommes **xmm?** (les registres disponibles vont **xmm0** a **xmm7**) qui ont une taille de **128bits** (16 octets)
- **AVX**: **Advanced Vector eXtensions** qui opere sur des vecteurs **ymm?** (**ymm0** a **ymm15**) qui ont une taille de **256bits** (32 octets) (SandyBridge)
- **AVX2**: **Advanced Vector eXtensions** qui opere sur des vecteurs **ymm?** (**ymm0** a **ymm15**) qui ont une taille de **256bits** (32 octets) ()
- **AVX512**: Similaire a AVX mais avec des registres **zmm?** (**zmm0** a **zmm31**) qui ont une taille de **512bits** (64 octets) - ce jeu d'instruction n'est disponible que sur les versions serveur d'Intel

# 2 Travail a effectuer

## 2.1 Code

Vous devez implementer plusieurs versions du code **dotprod** ci-dessous en deroulant autant de fois que vous desirez le corps de la boucle interne et comparer les codes assembleurs produits par le compilateur pour les differentes versions avec differentes options de compilation destinees a **produire du code assembleur vectoriel**.

Exemple du code non deroule:

```
double dotprod(double *restrict a, double *restrict b, unsigned long long n)
{
    double d = 0.0;

    for (unsigned long long i = 0; i < n; i++)
       d += a[i] * b[i];
```

```
    return d;
}
```

Exemple du code deroule 2 fois:

```
double dotprod_unroll2(double *restrict a, double *restrict b, unsigned long long n)
{
    double d1 = 0.0;
    double d2 = 0.0;

    for (unsigned long long i = 0; i < n; i += 2)
       {
  d1 += (a[i]     * b[i]);
  d2 += (a[i + 1] * b[i + 1]);
       }

    return (d1 + d2);
}
```

### 2.2 Options de compilation

Vous pouvez utiliser un ou plusieurs compilateurs, par exemple:

- GNU C Compiler **gcc**
- Intel C Compiler **icc**
- LLVM **clang**
- AMD Optimizing C Compiler **aocc**.

Il faudra compiler le programme en suivant les etapes suivantes (pour **gcc** et **clang**):

- Version de base: -O1
- Version legerement optimisee: -O2
- Version fortement optimisee 1: -O3

- Version fortement optimisee 2: -Ofast
- Version kamikaze: -march=native -mtune=native -Ofast -funroll-loops -finline-functions -ftree-vectorize

Vous pouvez utiliser d'autres options de compilation: **RTFM**.

Pour generer du code assembleur utilisez l'option: **-S** avec tous les compilateurs

# 3 Rendu

Vous devez fournir une archive contenant les codes sources et les codes assembleurs analyses en plus d'un **README** qui fournit des explications pour chacune des versions de la fonction et si les instructions SIMD x86 (jeux d'instructions SSE et AVX) ont ete utilisees de facon scalaire (i.e. add[sd] ou add[ss]) ou vectorielle (i.e. add[pd] ou add[ps]).

- [ss | sd]: **ss** pour **scalar simple** precision et **sd** pour **scalar double** precision (operations scalaires)
- [ps | pd]: **ps** pour **packed simple** precision et **pd** pour **packed double** precision (operations vectorielles)

Il est preferable de creer un **makefile** avec une regle par version (voir l'exemple fournit).

Il faudra aussi fournir un fichier contenant la version du compilateur utilise:

```
#Pour GCC
$ gcc --version > gcc_ver.txt

#Pour clang
$ clang --version > clang_ver.txt
```

Et un fichier contenant les informations sur votre processeur:

```
$ cat /proc/cpuinfo > cpuinfo.txt
```

Si vous decidez de faire tout le TD sur Compiler Explorer [1], vous n'avez pas besoin de fournir de **makefile**. Mais il faudra fournir les versions des compilateurs utilises et les micro-architectures cibles [4] (-march=ARCH) et utiliser au minimum les trois compilateurs suivants: gcc, clang et icc.

# 4 References:


- 0. RTFM (Read The Fucking Manual): **man gcc** ou **man clang** sur la console.
- 1. Compiler explorer: `https://gcc.godbolt.org/`
- 2. Intel Intrinsics Guide: `https://software.intel.com/sites/landingpage/IntrinsicsGuide/`
- 3. GCC optimization options: `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`
- 4. GCC x86 architectures: `https://gcc.gnu.org/onlinedocs/gcc/x86-Options.html`
- 5. Felix Cloutier's x86 instructions reference guide: `https://www.felixcloutier.com/x86/`
- 6. Agner Fog's optimization manuals: `https://www.agner.org/optimize/`