

# HPC projet 1 - Optimisation librairie Piccante

Edoardo Carpita

## Introduction

Pour mon projet personnel j'ai choisi de travailler avec une librairie de traitement d'image qui s'appelle Piccante. Cette librairie écrite en C/C++ permet d'effectuer des traitements d'images avancés.

Dans le cadre spécifique de ce projet je vais me concentrer sur une application en particulier, le generateur d'image HDR.

L'imagerie HDR consiste à travailler avec une suite de photos du meme sujet à des niveaux d'expositions différents, pour ensuite effectuer une fusion de l'image avec les pixels de meilleure qualité selon l'algorithme.

Justement parce que j'étais curieux de decouvrir plus sur cette technique dela photographie, j'ai décidé de travailler sur ce sujet.

J'ai commencé mon travail avec la creation d'un projet Clion avec comme base un programme de test qui permet de fusionner une image de test fournie par les developpeurs de la librairie.

La librairie etant composé principalement de fichiers hpp, l'integration dans mon projet se fait assez facilement. Une fois resolu des problèmes de compilation et adaté correctement le CMakeList.txt, j arrive à compiler et executer le programme une fois.

Dans mon travail d'optimisation je vais viser un travail "generaliste" (une serie d'images de dimension normales, ni trop grande ou trop petite), et je vise le maximum de performance par rapport à ma machine de travail.

Le programme se lance tout simplement apres la compilation via la commande `./hdr_generator` et les images dans `data/input` sont converties e une seule image HDR dans le dossier `data/output`

Le repos du projet se trouve ici [https://github.com/ecarpita93/HPC\\_projet\\_1](https://github.com/ecarpita93/HPC_projet_1)

## Situation baseline

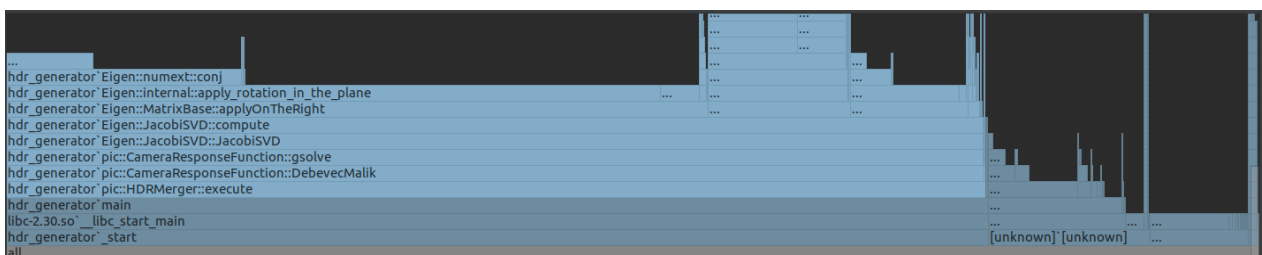
La première execution du programme, j'utilise les outils vus au cours afin d'effectuer par la suite des comparaisons sur le travail d'optimisation:

### Time

```
time ./hdr_generator

real    1m47.063s
user    1m47.601s
sys     0m0.028s
```

### Perf et flamegraph



```
edd993@edd993-ThinkPad-W530: ~/CLionProjects/hdr_generator/cmake-build-debug
Samples: 18K of event 'cycles', Event count (approx.): 685533749781
Overhead Command Shared Object Symbol
29.67% hdr_generator hdr_generator [.] Eigen::internal::apply_rotation_in_the_plane<Eigen::Block<Eigen::Matrix<float, -1, -1, 0, -1, -1>, -1, 1, true>, Eigen::Block<Eigen::Matrix<float, -1, -1, 0, -1, -1>, -1, 1, true>>::run
16.24% hdr_generator hdr_generator [.] Eigen::internal::conj_impl<float, false>::run
14.69% hdr_generator hdr_generator [.] Eigen::numext::conj<float>
6.97% hdr_generator [kernel.kallsyms] [k] nmi
4.37% hdr_generator [kernel.kallsyms] [k] nmi
3.92% hdr_generator hdr_generator [.] Eigen::internal::gebp_traits<float, float, false, false>::madd<float __vector(4), float __vector(4), float __vector(4)>
2.29% hdr_generator hdr_generator [.] Eigen::internal::pmul<float __vector(4)>
2.26% hdr_generator hdr_generator [.] Eigen::internal::padd<float __vector(4)>
2.14% hdr_generator hdr_generator [.] Eigen::internal::gebp_kernel<float, float, long, Eigen::internal::blas_data_mapper<float, long, 0, 0>, 8, 4, false, false>::operator(
1.46% hdr_generator hdr_generator [.] Eigen::internal::sub_assign_op<float, float>::assignPacket<16, float __vector(4)>
1.13% hdr_generator hdr_generator [.] Eigen::internal::pstore<float, float __vector(4)>
1.12% hdr_generator hdr_generator [.] Eigen::internal::general_matrix_vector_product<long, float, Eigen::internal::const_blas_data_mapper<float, long, 1>, 1, false, float, Eigen::internal::generic_dense_assignment_kernel<Eigen::internal::evaluator<Eigen::Block<Eigen::Block<Eigen::Block<Eigen::Matrix<float, -1, -1, 0, -1, -1>, -1, 1, true>, Eigen::Block<Eigen::Matrix<float, -1, -1, 0, -1, -1>, -1, 1, true>>::run
0.80% hdr_generator hdr_generator [.] Eigen::internal::padd<float __vector(4)>
0.80% hdr_generator hdr_generator [.] Eigen::internal::palign<2, float __vector(4)>
0.71% hdr_generator hdr_generator [.] Eigen::Matrix<float, -1, -1, 0, -1, -1>::innerStride
0.61% hdr_generator hdr_generator [.] Eigen::internal::palign<1, float __vector(4)>
0.58% hdr_generator hdr_generator [.] Eigen::internal::evaluator<Eigen::PlainObjectBase<Eigen::Matrix<float, -1, 1, 0, -1, 1> > >::packet<0, float __vector(4)>
0.55% hdr_generator hdr_generator [.] Eigen::internal::pload<float __vector(4)>
0.49% hdr_generator hdr_generator [.] Eigen::internal::scalar_product_op<float, float>::packetOp<float __vector(4)>
0.44% hdr_generator hdr_generator [.] Eigen::internal::gebp_traits<float, float, false, false>::loadLhs<float __vector(4)>
0.41% hdr_generator hdr_generator [.] Eigen::internal::palign_impl<1, float __vector(4)>::run
0.35% hdr_generator hdr_generator [.] Eigen::internal::palign<3, float __vector(4)>
0.33% hdr_generator hdr_generator [.] Eigen::internal::psub<float __vector(4)>
0.33% hdr_generator hdr_generator [.] Eigen::internal::pbroadcast<float __vector(4)>
0.32% hdr_generator hdr_generator [.] Eigen::internal::binary_evaluator<Eigen::CwiseBinaryOp<Eigen::internal::scalar_product_op<float, float>, Eigen::CwiseNullaryOp<Eigen::internal::scalar_constant_op<float>, true, false, false>::packetOp<float __vector(4)>, Eigen::Block<Eigen::Block<Eigen::Block<Eigen::Matrix<float, -1, -1, 0, -1, -1>, -1, 1, true>, Eigen::Block<Eigen::Matrix<float, -1, -1, 0, -1, -1>, -1, 1, true>>::run
0.29% hdr_generator hdr_generator [.] Eigen::internal::pset1<float __vector(4)>
0.29% hdr_generator hdr_generator [.] Eigen::DenseCoeffsBase<Eigen::Block<Eigen::Matrix<float, -1, -1, 0, -1, -1>, 1, -1, false>, 3>::innerStride
0.29% hdr_generator hdr_generator [.] Eigen::internal::conj_helper<float __vector(4), float __vector(4), false, false>::pmul
0.28% hdr_generator hdr_generator [.] Eigen::internal::evaluator<Eigen::CwiseNullaryOp<Eigen::internal::scalar_constant_op<float>, Eigen::Matrix<float, -1, 1, 0, -1, 1> >::run
0.23% hdr_generator hdr_generator [.] Eigen::internal::pbase_evaluator<Eigen::Block<Eigen::Block<Eigen::Block<Eigen::Matrix<float, -1, -1, 0, -1, -1>, -1, 1, false>, -1, -1, false>, -1, -1, false>>::run
0.23% hdr_generator hdr_generator [.] Eigen::internal::pload<float __vector(4)>
0.21% hdr_generator hdr_generator [.] Eigen::internal::scalar_constant_op<float>::packetOp<float __vector(4)>
0.21% hdr_generator hdr_generator [.] Eigen::internal::dense_assignment_loop<Eigen::internal::generic_dense_assignment_kernel<Eigen::internal::evaluator<Eigen::Block<Eigen::Block<Eigen::Block<Eigen::Matrix<float, 2, 2, 0, 2> >::operator, Eigen::CommaInitializer<Eigen::Matrix<float, 2, 2, 0, 2> >::operator, Eigen::CwiseNullaryOp<Eigen::internal::scalar_constant_op<float>, Eigen::Matrix<float, -1, 1, 0, -1, 1> >::run
0.17% hdr_generator [kernel.kallsyms] [k] __cgroup_account_cputime_field
0.16% hdr_generator hdr_generator [.] Eigen::SVDBase<Eigen::JacobiSVD<Eigen::Matrix<float, -1, -1, 0, -1, -1>, > >::computeV
0.15% hdr_generator hdr_generator [.] Eigen::internal::general_matrix_vector_product<long, float, Eigen::internal::const_blas_data_mapper<float, long, 1>, 1, false, float, Eigen::PlainObjectBase<Eigen::Matrix<float, -1, -1, 0, -1, -1> >::cols
Cannot load tips.txt file, please install perf!
```

En faisant des recherches, je comprends mieux le fonctionnement du programme et la fonctions des différentes fonctions que je peux observer sur la stack du flamegraph. EN bref, afin de générer une image HDR, et donc fusionner les 7 exemplaires à disposition en une seule image optimisée, il est nécessaire de connaître la "CRF" (Camera response function), qui représente la relation entre la quantité de lumière entrante et les valeurs correspondantes des pixels d'une image de la caméra. Cette valeur est estimable à partir de la séquence d'images d'exposition (car les compagnies des caméras considèrent cette information commerciale et non-divulgable), et pour faire cela un algorithme a été mis en place par les chercheurs Debevec et Malik. Cet algorithme par contre n'est pas complètement efficace sans corriger les effets de bruits parasites qui se produisent par échantillonnage (comme les images peuvent contenir des aberrations qui polluent le CRF), et la solution à ce problème a été trouvée dans la régularisation des données et cela s'effectue utilisant un calcul de décomposition en valeurs singulières. Comme on peut observer des données ci-dessus, la plus part de temps et de ressources sont dédiés dans l'application de cette régularisation des données, mon travail sera donc celui de trouver une façon d'optimiser si possible cette partie du code

# Optimisation 1: compilateur

## Application des optimisations de base:

En ayant créé moi-même le projet et la bibliothèque étant basée sur les headers, aucune optimisation au niveau du compilateur est présente. Je procède donc à rajouter à mon CMakeList des flags afin d'améliorer le travail de compilation du programme. Je teste plusieurs solutions, dont je trouve la plus performante dans mon cas avec cette configuration:

- *Ofast* : optimisation générale de la compilation la plus efficace que j'ai pu tester, elle permet de réduire le temps d'exécution de presque 2 minutes à environ 9 secondes
- *march=native* : optimisation qui indique au compilateur d'optimiser le code par rapport à la machine native ou la compilation a été effectuée. Native demande une version de gcc assez récente (gcc 5.4 et plus) mais non seulement ceci permet un gain dans la rapidité de l'exécution du programme mais permet aussi une approche plus portable que expliciter l'architecture de ma machine. En plus elle va s'occuper d'activer tous les instructions que mon ordinateur est capable de supporter sans devoir les expliciter (donc à priori ceci va activer la vectorisation du code déjà présente dans la bibliothèque).
- *ffast-math* et *freet-vectorize* : optimisation finale, permet de gagner un peu de temps cpu

Déjà sans outils d'analyse je remarque que le programme s'exécute beaucoup plus rapidement. Avec la commande `time` je peux reporter les temps effectifs d'exécution avec `time`:

real	0m8.616s
user	0m8.768s

sys 0m0.048s

## Application des optimisations par profiling automatique:

J'essaie de pousser au maximum la possibilité d'optimisation avec compilateur et j'effectue une nouvelle optimisation avec *-fprofile* afin de générer automatiquement une optimisation sur mesure avec l'outil de profiling intégré au compilateur.

Le profiling automatique s'avère une solution fonctionnelle et réduit encore un petit peu le temps d'exécution du programme.

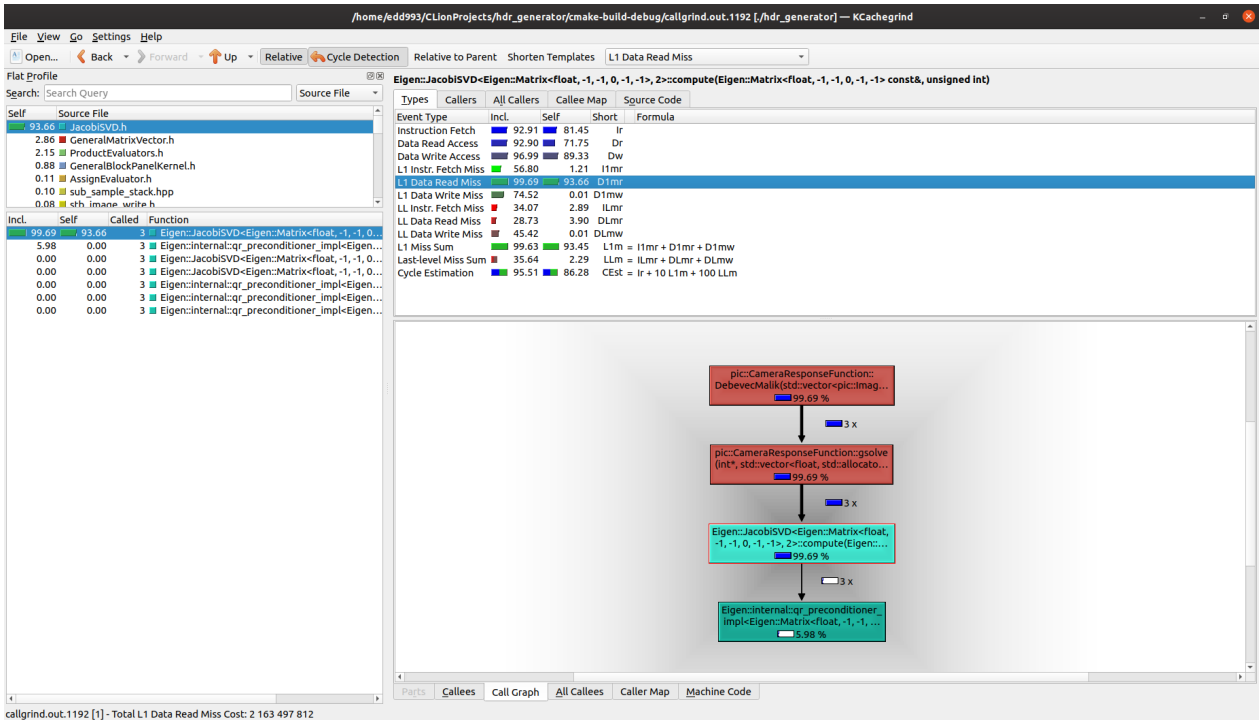
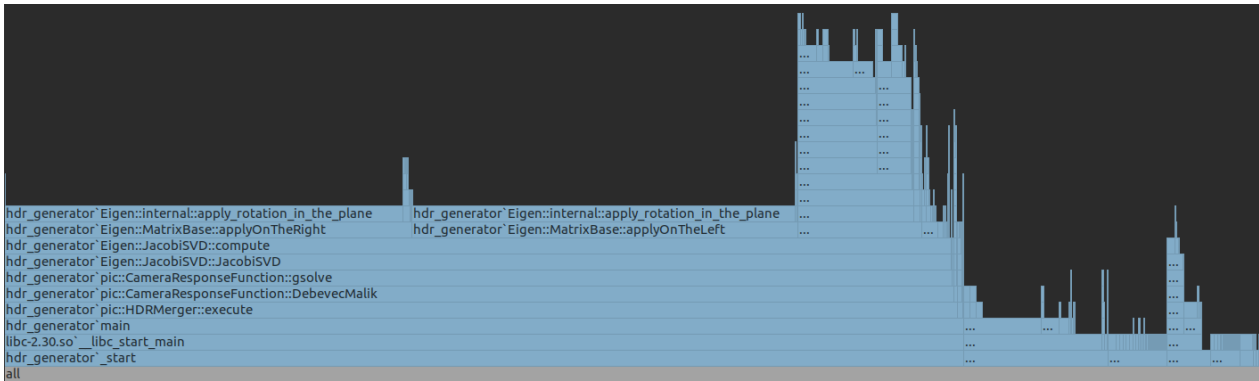
J'obtiens avec *time* le temps suivant:

```
real    0m8.219s
user    0m8.590s
sys     0m0.028s
```

## Nouveau profiling manuel

A ce point là je retiens sage de réeffectuer un autre fois les outils de profiling, comme la compilation optimisée a sûrement engendré des effets sur l'utilisation des ressources du programme. Une nouvelle exécution de *perf* et *valgrind* me pointe vers les points critiques du programme. (Pour cette partie je désactive l'inlining de l'optimisation avec *-fno-inline* afin de mieux observer les fonctions effectives qui consomment plus de ressources)

```
edd993@edd993-ThinkPad-W530: ~/CLionProjects/hdr_generator/cmake-build-debug
Samples: 40K of event 'cycles:u', Event count (approx.): 33341638828
Overhead Command Shared Object Symbol
29.93% hdr_generator hdr_generator [.] Eigen::internal::apply_rotation_in_the_plane<Eigen::Block<Eigen::Matrix<float, -1, -1, 0, -1, -1>, 1, -1, 1>
29.15% hdr_generator hdr_generator [.] Eigen::internal::apply_rotation_in_the_plane<Eigen::Block<Eigen::Matrix<float, -1, -1, 0, -1, -1>, -1, -1, 1
5.90% hdr_generator hdr_generator [.] Eigen::internal::gebp_traits<float, float, false, false>::madd<float __vector(8), float __vector(8), flo
2.49% hdr_generator hdr_generator [.] Eigen::internal::pmul<float __vector(8)>
2.24% hdr_generator hdr_generator [.] Eigen::internal::gebp_kernel<float, float, long, Eigen::internal::blas_data_mapper<float, long, 0, 0>, 1
1.92% hdr_generator hdr_generator [.] Eigen::internal::generic_dense_assignment_kernel<Eigen::internal::evaluator<Eigen::Block<Eigen::Block<Ei
1.58% hdr_generator hdr_generator [.] Eigen::internal::sub_assign_op<float, float>::assignPacket<32, float __vector(8)>
1.50% hdr_generator hdr_generator [.] Eigen::internal::pmadd<float __vector(8)>
1.41% hdr_generator hdr_generator [.] Eigen::internal::pbroadcast4<float __vector(8)>
1.33% hdr_generator hdr_generator [.] Eigen::internal::binary_evaluator<Eigen::CwiseBinaryOp<Eigen::internal::scalar_product_op<float, float>,
1.05% hdr_generator hdr_generator [.] Eigen::internal::gebp_traits<float, float, false, false>::loadLhs<float __vector(8)>
0.86% hdr_generator hdr_generator [.] Eigen::internal::padd<float __vector(8)>
0.85% hdr_generator hdr_generator [.] Eigen::internal::ploadu<float __vector(8)>
0.79% hdr_generator hdr_generator [.] Eigen::internal::general_matrix_vector_product<long, float, Eigen::internal::const_blas_data_mapper<fla
0.75% hdr_generator hdr_generator [.] Eigen::internal::conj_helper<float __vector(8), float __vector(8), false, false>::pmul
0.71% hdr_generator hdr_generator [.] Eigen::internal::mapbase_evaluator<Eigen::Block<Eigen::Block<Eigen::Block<Eigen::Matrix<float, -1, -1, 0
0.66% hdr_generator hdr_generator [.] pic::FilterAssembleHDR::ProcessBBox
0.54% hdr_generator libgomp.so.1.0.0 [.] 0x0000000000001dcb0
0.51% hdr_generator [unknown] [k] 0xffffffff8e200ae7
0.51% hdr_generator hdr_generator [.] Eigen::internal::pload1<float __vector(8)>
0.49% hdr_generator hdr_generator [.] Eigen::JacobiRotation<float>::transpose
0.40% hdr_generator libgomp.so.1.0.0 [.] 0x0000000000001dcb2
0.38% hdr_generator hdr_generator [.] Eigen::internal::pload<float __vector(8)>
0.37% hdr_generator hdr_generator [.] pic::CameraResponseFunction::remove
0.36% hdr_generator hdr_generator [.] Eigen::internal::BlockImpl_dense<Eigen::Matrix<float, -1, -1, 0, -1, -1>, -1, 1, true, true>::BlockImpl_
0.33% hdr_generator hdr_generator [.] Eigen::JacobiSVD<Eigen::Matrix<float, -1, -1, 0, -1, -1>, 2>::compute
0.27% hdr_generator hdr_generator [.] Eigen::internal::dense_assignment_loop<Eigen::internal::generic_dense_assignment_kernel<Eigen::internal:
0.25% hdr_generator hdr_generator [.] Eigen::internal::real_2x2_jacobi_svd<Eigen::Matrix<float, -1, -1, 0, -1, -1>, float, long>
0.25% hdr_generator libgomp.so.1.0.0 [.] 0x0000000000001dcb5
0.23% hdr_generator hdr_generator [.] stbi_zlib_compress
0.22% hdr_generator hdr_generator [.] Eigen::MapBase<Eigen::Block<Eigen::Matrix<float, -1, -1, 0, -1, -1>, -1, 1, true>, 0>::MapBase
0.22% hdr_generator libgomp.so.1.0.0 [.] 0x0000000000001de78
0.21% hdr_generator hdr_generator [.] Eigen::internal::BlockImpl_dense<Eigen::Block<Eigen::Map<Eigen::Matrix<float, -1, -1, 1, -1, -1> const,
0.21% hdr_generator hdr_generator [.] Eigen::PlainObjectBase<Eigen::Matrix<float, -1, -1, 0, -1, -1> >::coeff
0.20% hdr_generator hdr_generator [.] Eigen::internal::variable_if_dynamic<long, 1>::value
0.19% hdr_generator hdr_generator [.] stbiw_zlib_countm
0.19% hdr_generator hdr_generator [.] Eigen::internal::pstore<float, float __vector(8)>
0.19% hdr_generator hdr_generator [.] Eigen::EigenBase<Eigen::Matrix<float, -1, -1, 0, -1, -1> >::rows
0.18% hdr_generator hdr_generator [.] Eigen::internal::scalar_constant_op<float>::packetOp<float __vector(8)>
0.16% hdr_generator hdr_generator [.] Eigen::DenseBase<Eigen::Matrix<float, -1, -1, 0, -1, -1> >::col
0.16% hdr_generator hdr_generator [.] Eigen::JacobiRotation<float>::makeJacobi
Cannot load tips.txt file, please install perf!
```



Dans ce cas on peut observer que la fonction qui consomme pratiquement toute seule le 7/10 du temps cpu du programme et aussi le plus haut taux de cache read miss est la fonction *apply\_rotation\_in\_the\_plane*, qui est appelé dans les fonctions supérieures *rotation\_right* et *rotation\_left*. Ma prochaine etape ca sera donc celle d'etudier un plan d'attaque pour cette fonction pour verifier si c'est possible de l'optimiser.

## Optimisation 2: parallelisation et vectorisation du code critique

### Tentative par modification du code directe

Le code de la fonction critique se trouve donc dans le dossier `include/external/Eigen/src/Jacobi` et effectue la rotation des données de la matrice utilisé pour le calcul de la decomposition en valeur singulières:

```
void /*EIGEN_DONT_INLINE*/ apply_rotation_in_the_plane(DenseBase<VectorX>& xpr_x, DenseBase<VectorY>& xpr_y,
{
    typedef typename VectorX::Scalar Scalar;
    enum {
        PacketSize = packet_traits<Scalar>::size,
        OtherPacketSize = packet_traits<OtherScalar>::size
    };
};
```

```

typedef typename packet_traits<Scalar>::type Packet;
typedef typename packet_traits<OtherScalar>::type OtherPacket;
eigen_assert(xpr_x.size() == xpr_y.size());
Index size = xpr_x.size();
Index incrx = xpr_x.derived().innerStride();
Index incry = xpr_y.derived().innerStride();

Scalar* EIGEN_RESTRICT x = &xpr_x.derived().coeffRef(0);
Scalar* EIGEN_RESTRICT y = &xpr_y.derived().coeffRef(0);

OtherScalar c = j.c();
OtherScalar s = j.s();
if (c==OtherScalar(1) && s==OtherScalar(0))
    return;

/** dynamic-size vectorized paths */
if(VectorX::SizeAtCompileTime == Dynamic &&
    (VectorX::Flags & VectorY::Flags & PacketAccessBit) &&
    (PacketSize == OtherPacketSize) &&
    ((incrx==1 && incry==1) || PacketSize == 1))
{
    // both vectors are sequentially stored in memory => vectorization
    enum { Peeling = 2 };

    Index alignedStart = internal::first_default_aligned(y, size);
    Index alignedEnd = alignedStart + ((size-alignedStart)/PacketSize)*PacketSize;

    const OtherPacket pc = pset1<OtherPacket>(c);
    const OtherPacket ps = pset1<OtherPacket>(s);
    conj_helper<OtherPacket, Packet, NumTraits<OtherScalar>::IsComplex, false> pcj;
    conj_helper<OtherPacket, Packet, false, false> pm;

    for(Index i=0; i<alignedStart; ++i)
    {
        Scalar xi = x[i];
        Scalar yi = y[i];
        x[i] = c * xi + numext::conj(s) * yi;
        y[i] = -s * xi + numext::conj(c) * yi;
    }

    Scalar* EIGEN_RESTRICT px = x + alignedStart;
    Scalar* EIGEN_RESTRICT py = y + alignedStart;

    if(internal::first_default_aligned(x, size)==alignedStart)
    {
        for(Index i=alignedStart; i<alignedEnd; i+=PacketSize)
        {
            Packet xi = pload<Packet>(px);
            Packet yi = pload<Packet>(py);
            pstore(px, padd(pm.pmul(pc,xi),pcj.pmul(ps,yi)));
            pstore(py, psub(pcj.pmul(pc,yi),pm.pmul(ps,xi)));
            px += PacketSize;
            py += PacketSize;
        }
    }
    else
    {
        Index peelingEnd = alignedStart + ((size-alignedStart)/(Peeling*PacketSize))*(Peeling*PacketSize);
        for(Index i=alignedStart; i<peelingEnd; i+=Peeling*PacketSize)
        {
            Packet xi = ploadu<Packet>(px);
            Packet xi1 = ploadu<Packet>(px+PacketSize);
            Packet yi = pload <Packet>(py);
            Packet yi1 = pload <Packet>(py+PacketSize);
            pstoreu(px, padd(pm.pmul(pc,xi),pcj.pmul(ps,yi)));
            pstoreu(px+PacketSize, padd(pm.pmul(pc,xi1),pcj.pmul(ps,yi1)));
            pstore (py, psub(pcj.pmul(pc,yi),pm.pmul(ps,xi)));
            pstore (py+PacketSize, psub(pcj.pmul(pc,yi1),pm.pmul(ps,xi1)));
            px += Peeling*PacketSize;
            py += Peeling*PacketSize;
        }
        if(alignedEnd!=peelingEnd)
        {
            Packet xi = ploadu<Packet>(x+peelingEnd);

```

```

        Packet yi = pload<Packet>(y+peelingEnd);
        pstoreu(x+peelingEnd, padd(pm.pmul(pc,xi),pcj.pmul(ps,yi)));
        pstore (y+peelingEnd, psub(pcj.pmul(pc,yi),pm.pmul(ps,xi)));
    }
}

for(Index i=alignedEnd; i<size; ++i)
{
    Scalar xi = x[i];
    Scalar yi = y[i];
    x[i] = c * xi + numext::conj(s) * yi;
    y[i] = -s * xi + numext::conj(c) * yi;
}
}

/** fixed-size vectorized path */
else if(VectorX::SizeAtCompileTime != Dynamic &&
        (VectorX::Flags & VectorY::Flags & PacketAccessBit) &&
        (PacketSize == OtherPacketSize) &&
        (EIGEN_PLAIN_ENUM_MIN(evaluator<VectorX>::Alignment, evaluator<VectorY>::Alignment)>0)) // FIXME sh
{
    const OtherPacket pc = pset1<OtherPacket>(c);
    const OtherPacket ps = pset1<OtherPacket>(s);
    conj_helper<OtherPacket,Packet,NumTraits<OtherPacket>::IsComplex,false> pcj;
    conj_helper<OtherPacket,Packet,false,false> pm;
    Scalar* EIGEN_RESTRICT px = x;
    Scalar* EIGEN_RESTRICT py = y;
    for(Index i=0; i<size; i+=PacketSize)
    {
        Packet xi = pload<Packet>(px);
        Packet yi = pload<Packet>(py);
        pstore(px, padd(pm.pmul(pc,xi),pcj.pmul(ps,yi)));
        pstore(py, psub(pcj.pmul(pc,yi),pm.pmul(ps,xi)));
        px += PacketSize;
        py += PacketSize;
    }
}

/** non-vectorized path */
else {
    for (Index i = 0; i < size; ++i) {

        Scalar xi = *x;
        Scalar yi = *y;
        *x = c * xi + numext::conj(s) * yi;
        *y = -s * xi + numext::conj(c) * yi;
        x += incrx;
        y += incry;
    }
}
}

```

Dans les lectures des données juste ici dessus, j'ai pu remarquer que la même fonction *apply\_rotation\_in\_the\_plane*, selon les données en paramètre des fonctions supérieures, utilise plus ou moins la même quantité de temps CPU. Donc on dirait que le temps d'exécution est assez constant même avec différentes données. Je aussi remarque à cette phase là que la vectorisation du programme, (qui contient comme on peut voir deux versions de l'algorithme déjà vectorisés) ne semble pas s'effectuer correctement, car les résultats des analyses avec perf annotate, valgrind, et même en debug avec breakpoint dans l'IDE j'obtiens des informations qui font seulement référence que à la version du code qui semblerait être "non-vectorized path". Cependant, toujours avec le debug de l'IDE qui me laisse pas ajouter des breakpoints à l'intérieur de la boucle et par analyse de code de perf avec annotate je remarque l'utilisation d'instructions SIMD dans le code, donc je suppose le compilateur assez capable de vectoriser tout seul cette partie de code.

```

else {
    for (Index i = 0; i < size; ++i) {
        Scalar xi = *x;
        Scalar yi = *y;
        *x = c * xi + numext::conj(s) * yi;
        *y = -s * xi + numext::conj(c) * yi;
        x += incrx;
        y += incry;
    }
}

```

Si on regarde coté parallélisation par thread, il ne me semblerait pas possible de paralléliser cet algorithme. Comme on peut observer dans le code, les valeurs de x et y sont mises à jour dans la boucle à chaque itérations, un peu comme la situation du laboratoire sur les fractales IFS.

## Tentative par automatisisation de la parallelisation

Au cours ultérieures recherches sur la parallelisation, je tombe sur la possibilité de j'essaie de paralléliser automatiquement avec l'option `-fopenmp-parallelize-loops=n` qui travaille aussi avec une version interne au compilateur de OpenMP. Ceci vas pas impacter surement la boucle ci dessus qui n'est pas parallelisable, mais peut etre peut quand meme engendrer un effet positif sur le programme et la modification etatn simple je pense ca vaut le coup d'essayer. Je modifie le CMake afin de compiler avec cette option et j'effectue différents tests (avec n=2,3,4,8). Ceci ne semble pas produire un effet positif sur le programme, et à partir d'une parallelisme avec 8 threads au contraire on obtiens une perte de performances considerable d'une seconde, surement du au couts de synchronisation trop élevés. Avec n=2 j'obtiens un léger gain, donc je decide de garder cette valeur dans le CMAKE

```

real      0m8.145s
user      0m8.524s
sys       0m0.035s

```

## Optimisation 4: remplacement de la librairie de calcul

Je decide d'applique un autre approche et je trouve que une nouvelle version de la librairie de calcul matriciel (Eigen) est disponible et que le programme prevoit un système afin d'utiliser une version non "bundled" de cette librairie. J'essaie donc de voire si cette nouvelle version disponible est plus efficace que la precedente dans les calculs. Afin de tester je telecharge donc la nouvelle version et remplace l'existante. Je teste la librairie et j'obtiens une petite amelioration des performances et avec l'outil de debug j'arrive à confirmer que la version optimisé du programme avec vectorisations est utilisé, bien que cela ne semble pas particulièrement influencer les temps d'execution du programme, ce qui me confirme ma theorie d'optimisation du compilateur du point precedent.

```

real      0m8.038s
user      0m8.462s
sys       0m0.041s

```

## Conclusions

J'estime le travail d'optimisation effectué pour ce projet dans le complexe pas très satisfaisant. Avec maintenant un regard en arriere sur le travail de recherche effectué et les benefices apportés à l'optimisation j'estime que mes recherches et mes test n'ont pas vraiment apporté une grande optimisation au programme, et ceci est du je pense à deux facteurs principaux:

- J'ai sous estimé la complexité de la librairie pendant la phase de recherche. bien que je trouve toujours le sujet très intéressant et j'ai appris beaucoup de notions très intéressantes sur la transposition HDR des images, le code est n'est pas particulièrement facile à comprendre avec l'utilisation masive des typenames et des templates c'est pas facile de savoir exactement ce qui fait le programme, et meme avec un IDE en debug j'ai trouvé des comportements bizarres (surememnt du à mon ignorance dans certaines manipulations du C++) qui m'ont empeché de comprendre en plein assez rapidement toute les composantes de la librairie afin dy pouvoir facilement appliquer des optimisations niveau codage directe. Je pense que meme si le sujet reste pour moi très intéressant, si je devais de nouveau choisir un programme à analyser je choisisserais un outils moins complexe car j'estime que le temps que j'ai pu dedier au projet n'as pas ete suffisant pour obtenir des resultats concrets (par exemple j'ai pas reussi à explorer dans le details les raisons des caches miss et si on pouvait les eviter).
- La librairie est aussi très bien entretenue, et souvent des methodes d'optimisations on ete deja mis en places par les developpeurs, comme l'utilisation de la vectorisation, ce qui ont rendu mon travail plus "interessant"