

RAPPORT DE STAGE

Stage du 16 mai 2022 au 15 juillet 2022

GOEDEFROIT Charles

11 juillet 2022

Table des matières

1 Contente

J'ai effectuer mon stage dans l'équipe STORM de Inria...
// petite description de Inria (labo...)
// description de l'équipe STORM (sigle, objectifs...)
// le personnels de l'équipe qui est constituer des permanents (chercheurs, enseignants chercheurs, ingé...),des doctorants et des stagiaires...
// culture informatique (chercher en info / étudiants en info)
// materiel et les logiciels a disposition :
// * un ordinateur portable avec linux et la possibilité d'installer ce que je veut en relation avec le travail bien-sûr
// * un accès à *PlaFRIM* (cluster de serveur avec plusieurs noeuds qui on des architecture différentes)...
// * ...

2 Objet du stage / travaille demandé

// faire le portage de Eigen sur MIPP
// Eigen est une lib... elle est capable de faire des calcule vectoriser mais pour cela elle a une implementation par architectures.
// MIPP est une lib... MIPP est un sous partie d'AFF3CT (A Fast Forward Error Correction Toolbox) qui...
// MIPP permet de faire des operations vectoriel...
// Le premier objectif est donc d'ajouter, avec l'interface MIPP, une implementation vectoriel des fonctions élémentaires de Eigen pour que le support des architectures sois automatique.
// Le second objectif est de faire une campagne d'évaluation des performances (avant / après MIPP)
// Le dernier objectif est de tester Eigen sur l'architecture Risc-V qui n'est pas encore présent den Eigen et évaluer les performances sur simulateur.
// L'objectif a long terme est de pouvoirs garder uniquement l'implementation MIPP et supprimer les autre.
// Un des intérêt de faire ce portage et de permettre le support de future architectures sans avoir a l'ajouter mais juste en mettant a jour MIPP.

3 Travaille réaliser

// J'ai regarder comment ajouter une architecture
// J'ai remarquer que dans chaque architecture il y a 4 fichier
// 1. PacketMath.h : qui définit les type Eigen et les operations (vectoriel et scalaire) sur les entier et les flottant de différent tailles.
// 2. TypeCasting.h : qui défini les conversion entre les différent type Eigen.
// 3. MathFunctions.h : qui définit les operations mathématique non élémentaire (comme : log, log2, log1p, expm1, exp, sin, cos, sqrt, rsqrt, reciprocal, tanh, frexp, ldexp)
// 4. Complex.h : qui définit les type vectoriel complex Eigen les operations sur les complex

```
// dans 'Eigen/src/Core/util/ConfigureVectorization.h' j'ai ajouter un cas
// MIPP
// j'ai ajouter une nouvelle architecture MIPP dans 'Eigen/src/Core/arch/'
// et que j'ai charger dans 'Eigen/Core'
// J'ai copier le contenu des architecture SSE, AVX et AVX512 dans les
// même fichier. C'est 3 architecture qui fonctionne ensemble (AVX depend de
// SSE, etc)
// J'ai listé les différent type Eigen définit pour chaque architecture aussi
// que les operations supporter (tableau avec toute les fonctions)
```

SSE	AVX	AVX2	AVX512
Packet4f	Packet8f	Packet4l	Packet16f
Packet2d	Packet4d		Packet8d
Packet4i	Packet8i		Packet16i
Packet16b	Packet8h		Packet16h
	Packet8bf		Packet16bf

TABLE 1 – Les type vectoriel Eigen par architecture

```
// J'ai lancer les tests Eigen sur mon architecture fraîchement copier de
// AVX.
// En lancent les tests Eigen j'ai remarquer qu'il sont très long à ce lancer
// et qu'il ne fonctionne pas à tous les coup.
// Pour que je puis tester efficacement et rapidement j'ai implémenter des
// tests de non regression. Pour exécuter ces tests j'ai copier les fonctions actuel
// dans un nouveaux fichier et j'ai suffixés ces fonctions par _old.
// J'ai implémenter les test de non regression pour la fonctionne pset1, qui
// rempli un vecteur avec la même valeur a chaque case, et j'ai fait l'implémentation
// en MIPP. Pour finir j'ai lancer mes tests qui passé.
// J'ai implémenter les test de non regression pour toutes les operations
// présent dans PacketMath.h. Ces test fonctionne pour les architectures SSE,
// AVX, AVX2 et quelque AVX512.
// Mes test de non régression son capable d'afficher le contenu des vecteur
// dans le cas ou il y a une différence entre la nouvelle version et l'ancienne version
// de l'operation. Il sont aussi capable d'afficher le contenu des vecteur en binaire
// et d'afficher sans plantage quant une operations n'existe pas dans MIPP.
// Pour faire fonctionné MIPP et Eigen ensemble j'ai regarder de plus prés
// comment il utilise les vecteurs :
```

3.1 La representation de vecteur dans Eigen

```
// Des vecteur à taille fix qui sont dépendent de l'architecture visé ...
// Il sont nommer Packet suivi d'un nombre, le nombre d'élément qu'il
// contient, et qui termine par une ou plusieurs letter qui corresponde qu type des
// éléments dans le vecteur. Par exemple un vecteur qui contient 4 flottant sera
// Packet4f.
// Eigen est capable d'utilise plusieurs tailles de vecteur à la fois...
```

3.2 La representation de vecteur dans MIPP

// Des vecteur avec des taille qui sont variable et indépendant de l'architecture.

Il y a donc que 2 type de vecteur *reg* et *reg2* (qui fait la moitié de la taille de *reg*).

// MIPP n'est pas capable d'utilisé plusieurs taille de vecteur à la fois.

// J'ai commencer sur une mauvaise piste con convertissant les Packet Eigen en *reg2* lors qu'il été petit. Ce qui fait que je devais faire la transformation, *combinate()*, entre *reg* et *reg2* a chaque fois que je voulais faire un calcule car, dans MIPP, il n'y a presque aucune operations sur les *reg2*. De plus cela ne fonctionné pas avec AVX512 et ajouter pleins de complication ce qui fait que j'ai du faire des conversion.

// Comme il MIPP ne peut pas utilisé des taille de vecteur différant je suis obliger de cast les vecteur entre Eigen et MIPP... je me retrouve donc à faite des conversion entre des vecteurs de taille différant par exemple un vecteur de 128 bits que je dois convertir en vecteur de 512 bits et inversement. Ces conversion peuvent amené a une perte de performance. (Benchmark here?)

// Dans un premier temps, pour chaque fonction, j'ai fait l'implémentation en MIPP pour chaque type quelle support. Ce qui ma amené à avoir 2 fois la même implémentation MIPP avec seulement le type qui change par exemple une *pset1<Packet4f>()* et une autre *pset1<Packet4i>*. Dans un second temps le but à été de *replier* les fonctions pour en avoir qu'une seul. Mais je suis tombés sur un problème, le comportement par défaut et une operation scalaire. En effet Eigen permet d'activer ou non la vectorisation. Lors que la vectorisation et désactiver se sont les même operations qui sont appelé mais, dans ce cas, c'est l'operation par défaut qui est appelé et quant la vectorisation et activer on tombe sur un cas spécifique (Le Packet) et preciser a l'appelle. On ce retrouve donc avec 2 cas spécifique, un pour les scalaire et un pour les vecteurs. Cette situation amené à un état compliqué qui reste encore à résoudre. Peut-être 2 solutions :

// * Tous coder et faire en sorte que tous les Packet corresponde au type *reg* de MIPP, ce qui permet d'avoir un seul type.

// * Ajouter des adaptateur pour avoir une arborescence et utiliser le polymorphisme.

// Grace à mes tests et mes quelque implémentation j'ai pu lister des operations qu'il manque dans MIPP :

× : absent de MIPP ✓ : present dans MIPP

	AVX512	AVX2	AVX	SSE4.1/4.2	SSE2/3
add<int32_t>	✓	✓	×	✓	✓
sub<int32_t>	✓	✓	×	✓	✓
mul<int32_t>	✓	✓	×	✓	×
orb<int8_t>	✓	✓	×	✓	✓
xor<int8_t>	✓	✓	×	✓	✓
and<int8_t>	✓	✓	×	✓	✓
cmpneq<int16_t>	×	✓	×	×	×
cmpneq<int8_t>	×	✓	×	×	×

TABLE 2 – Abstractions MIPP non implementer

```
// Pour exécuter le code en AVX512 j'ai utilisé PlaFRIM...
```

4 Conclusion

```
// J'ai remarquer que :  
// * Les test unitaire de Eigen sont très long ce qui fait qui ne mon pas été  
utile pour teseter mes modifications  
// * Les test unitaire de Eigen on des bug ce qu'il fait qu'il faut les lancer  
plusieurs fois pour qu'il fonctionne (c'est un bug connue) (trouver l'issus)  
// * J'ai remarquer que plus on code en MIPP dans Eigen plus le temps de  
compilation est élever. C'est très certainement du au templates mais je ne peut  
pas dire si cela viens des templates dans MIPP ou dans Eigen ou les 2.  
// Les connaissances que j'ai acquis lors de mes étude que j'ai utiliser au  
cours du stage :  
// * Les test de non regression que j'ai du implementer pour verifier que mes  
implementation été bonne. (AL / PdP / PLE / SLAM4) (Le GL)  
// * Le fait de refactorisé du code (utilisation des 5 principe...) (AL / PdP)  
// * L'utilisation des intransics et le calcule vectoriel (vue en PAP)  
// * Les base en C++ vue un petit-peu en OS dans nahos mais aussi dans  
mon projet PdP...  
// * La comprehension du monde de la recherche grace à l'UE initiation  
recherche.  
// Ce que ma apporter le stage :  
// * une mateur comprehension des cast, conversion des type de base et des  
vecteurs  
// * Utilisation avancer des templates en c++  
// * des Utilisation différentes des vecteurs, plusieurs vision de leur utiliza-  
tion. (taille fix ou non)  
// * une mateur comprehension du monde de la recherche
```

5 Future / ce qui reste à faire

```
// Il reste a implementer tous le reste  
// Faire un repliage qui fonctionne  
// Modifier MIPP pour ajouter tous ce qu'il manque...  
// Evolution possible pour MIPP :  
// * Ajout du type bool, d'un type h et complex (present en AVX512)  
// * ajouter la correspondance int8_t et char (pour le cast)  
// * ajout des type unsigned (uint8_t, uint16_t...)  
// * make Reg2 printable  
// * peut-être ajouter les operations pour le Reg2 mais ce n'ai peut-être pas  
dans l'esprit de MIPP  
// * faire en sorte que le testz fonctionne avec toute les architecture  
// * voire le temps de compilation
```

En **vert** les type qui fonctionne, En **orange** les type qui ne fonctionne

Type standard	type de base	type unsigned
int8_t	char	uint8_t
int16_t	short	uint16_t
int32_t	int	uint32_t
int64_t	long	uint64_t
float	—	—
double	—	—
bool	—	—

TABLE 3 – Type compatible avec MIPP

6 Remerciements

// merci