Université de Versailles
Saint-Quentin-en-Yvelines
Département de Informatique

Master 1 Calcul Haute Performance, Simulation

---

# *PPN- Loop Tiling*

---

*Réalisé par :*
- Bouchelga Abdeljalil

*Encadré par :*
- Dr. Hugo Bolloré

Année Universitaire :2021-2022

# TABLE DES MATIÈRES

# TABLE DES FIGURES

PARTIE 1

# LOOP TILING

## 1.1 Loop Tiling

Il s'avère que notre code n'utiliser pas la mémoire cache pour un accès plus rapide à la mémoire. Une solution à cela consiste à utiliser une technique appelée **Loop Tiling**.

Rappelons que l'architecture de nos machine actuelle et comme suivant :

- **Le cache L1** est de 32 Kio par coeur. C'est $32 * 1024 = 32,768$ bytes par core.
- **Le cache L2** est de 512 Kio par cœur (1 Mio par tuile = 1024 Ko par tuile. Chaque tuile a 2 cœurs. Donc 1024 KiB par tiles / 2 cores per tiles = 512 KiB par core), c'est $512 * 1024 = 524,288$ bytes par core.

Lorsque la taille de nos probleme est petit les données dans une boucle peuvent tenir dans le cache L1/L2 pour des performances élevées/moyennes. Lorsque la taille devient trop élevé, les données peuvent ne pas tenir dans le cache et "se répandre" dans la RAM et pas le cache, ce qui est beaucoup plus lent. Cela entraîne une dégradation des performances.

Il s'avère qu'en pratique, nous pouvons améliorer les performances de notre cache avec la technique **Loop Tiling**. C'est une technique conçue pour conserver votre jeu de donnes dans des caches pendant que nous travaillons avec, pour profiter de la latence de la mémoire.

l'image suivante est tirée de l'article [1]

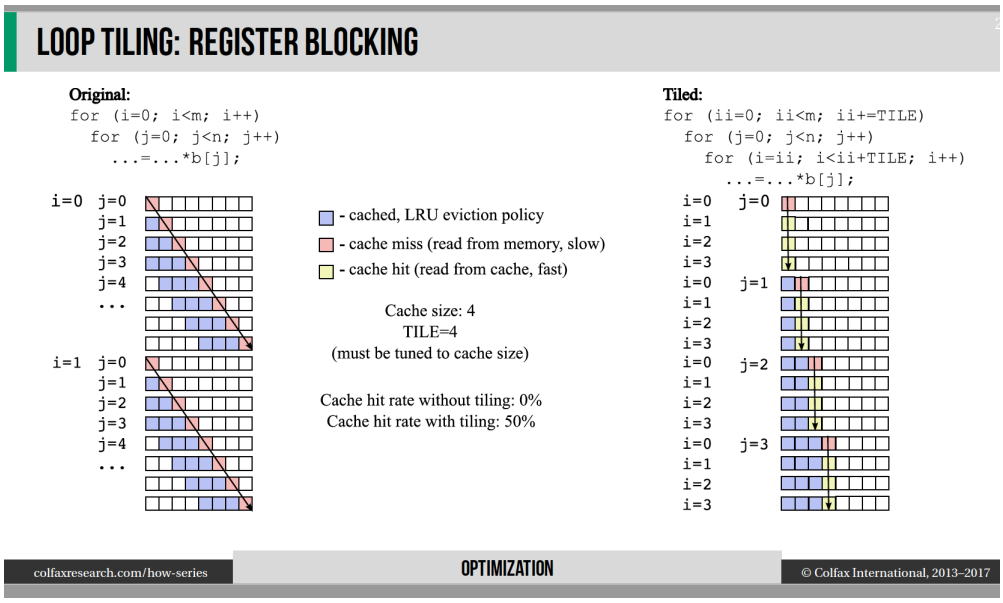


FIGURE 1.1 – Loop Tiling

Ce qui se passe en pratique, c'est que l'utilisation de loop tilling permet de réorganise les accès en blocs, de sorte que la répétition du même bloc peut atteindre le cache plusieurs fois. Étant donné que la taille du bloc est définie par exemple sur 32k, il s'adaptera probablement à la plupart des caches.Et cela peut facilement s'intégrer dans le cache L1 et on aura un taux d'accès au cache élevé (performance élevé) sinon si la taille est plus grand que 32,768, il se peut tenir dans le cache L2 (performance moyenne).

## Application

Exemple de la boucle qui fait la produit de convolution dans la fonction **convolution_process**

## Avant :

```
for (int ii = 0; ii < ConvMat_height; ii++) { //loop on the height of the
convolution matrix
        for (int jj = 0; jj < ConvMat_width; jj++) { //loop on the width of
the convolution matrix

      double sum = 0; //initialization of the  summation
         ...........
      double image = (pixel[((ii + kk) * (ConvMat_width + 2) + (jj + hh))]);
      sum += (image * filter_matrix[idx][kk * filter_width + hh]);
         .......
```

### Après :

```
for (int a = 0; a < ConvMat_height; a+=Tile) {
   for (int jj = 0; jj < ConvMat_width; jj++) { //loop on the width of the
convolution matrix
     for(int ii = a; ii< a+Tile; ii++){
      double sum = 0; //initialization of the  summation
            ...........
      double image = (pixel[((ii + kk) * (ConvMat_width + 2) + (jj + hh))]);
       sum += (image * filter_matrix[idx][kk * filter_width + hh]);
       .....
```

La syntaxe **après** produit le même résultat que la syntaxe **avant**, mais il le fait simplement d'une manière plus efficace pour la machine (en tirant le meilleur parti du cache rapide).

# BIBLIOGRAPHIE

[1] "Loop Tiling", https://huyenchip.com/2021/09/07/a-friendly-introduction-to-machine-learn
html