

UNIVERSITÉ DE VERSAILLES
SAINT-QUENTIN-EN-YVELINES
DÉPARTEMENT DE INFORMATIQUE

MASTER 1 CALCUL HAUTE PERFORMANCE, SIMULATION

PPN- Loop Tiling - Loop unrolling

Réalisé par :

- BOUCHELGA ABDELJALIL

Encadré par :

- DR. HUGO BOLLORÉ

Année Universitaire :2021-2022

TABLE DES MATIÈRES

List of figures	2
1 Introduction	3
1.1 Loop Tiling	3
1.2 Déroulement de la boucle (Loop unrolling)	5
Bibliographie	6

TABLE DES FIGURES

1.1	Loop Tiling	4
-----	-----------------------	---

1.1 Loop Tiling

Il s'avère que notre code n'utilise 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 machines actuelles est comme suit :

- **Le cache L1** est de 32 Kio par cœur. 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 problèmes est petite les données dans une boucle peuvent tenir dans le cache L1/L2 pour des performances élevées/moyennes. Lorsque la taille devient trop élevée, 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 données dans des caches pendant que nous travaillons avec, pour profiter de la latence de la mémoire.

L'image suivante est tirée de l'article

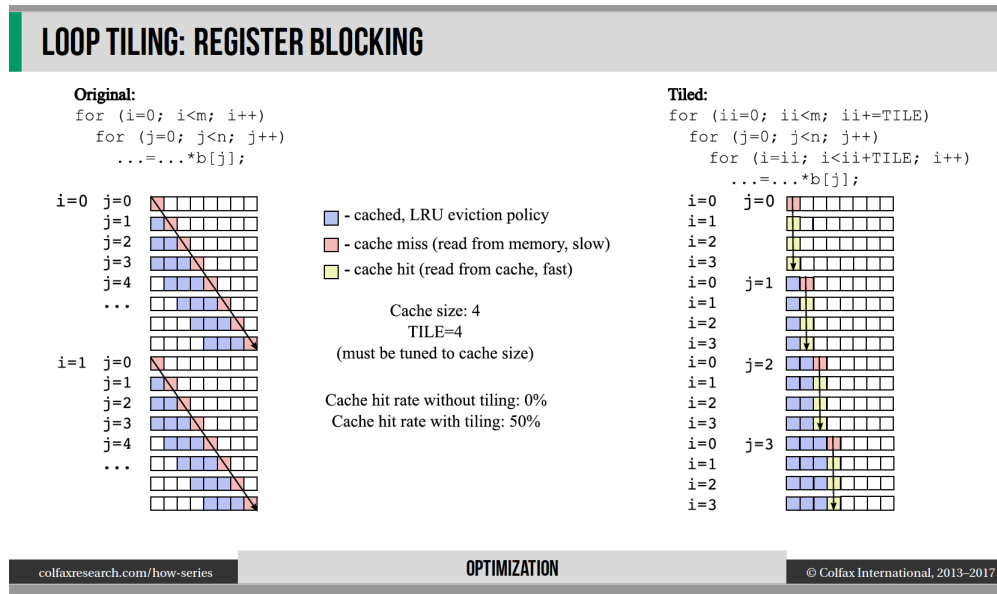


FIGURE 1.1 – Loop Tiling

Ce qui se passe en pratique, c'est que l'utilisation de loop tiling permet de réorganiser 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 élevée) sinon si la taille est plus grande 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).

1.2 Déroulement de la boucle (Loop unrolling)

Le déroulement de boucle est une technique de transformation de boucle qui permet d'optimiser le temps d'exécution d'un programme. Nous supprimons ou réduisons essentiellement les itérations. Le déroulement de la boucle augmente la vitesse du programme en éliminant les instructions de contrôle de boucle et les instructions de test de boucle.

La bonne chose à propos de la version déroulée est qu'elle implique moins de charge de traitement pour le processeur.

Nous allons ajouter le drapeau **#pragma unroll** aux boucles qui permet au compilateur de "dérouler" des boucles.

BIBLIOGRAPHIE

- [1] "Scaling tutorial", https://hpc-wiki.info/hpc/Scaling_tutorial
- [2] "Scalabilité", <https://hmf.enseeiht.fr/travaux/projnum/2020/programmation-parall%C3%A8le-fortran-avec-mpi/passage-%C3%A0-l%e2%80%99%C3%A9chelle-%e2%80%93-scalabilit%C3%A9>
- [3] "Scalability : strong and weak scaling", <https://bisqwit.iki.fi/story/howto/openmp/>