Optimisation du Code C - Rapport

Reyad Ouahi

February 20, 2025

1 Introduction

Dans ce rapport, nous présentons les optimisations effectuées sur une fonction C initiale. Ces optimisations visent à améliorer la performance du programme en réduisant le nombre d'opérations inutiles et en améliorant l'accès à la mémoire.

2 Code Initial

Le code initial est présenté ci-dessous :

Listing 1: Code initial

```
void kernel (unsigned n, float a[n], float b[n], float c[n][n]) {
    unsigned i, j;
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
            a[i] += c[i][j] / b[i];
        }
    }
}
```

3 Problèmes Identifiés

Plusieurs problèmes de performance ont été identifiés dans ce code :

- **Ordre des boucles** : L'accès mémoire à 'c[i][j]' suit un ordre *colonne-major*, ce qui entraîne des accès inefficaces en mémoire cache.
- **Opérations redondantes** : L'opération 'c[i][j] / b[i]' est répétée à chaque itération, augmentant le nombre d'opérations de division coûteuses.
- **Absence de vectorisation et d'optimisation du pipeline** : Le compilateur ne peut pas facilement vectoriser le code, limitant ainsi les gains de performance.

4 Optimisations Apportées

Nous avons apporté plusieurs modifications pour améliorer l'efficacité du code.

4.1 Réorganisation des boucles

Nous avons inversé l'ordre des boucles pour un accès mémoire optimisé, réduisant ainsi le nombre de cache misses.

4.2 Élimination des divisions redondantes

L'opération de division a été déplacée en dehors de la boucle interne pour éviter de la recalculer à chaque itération.

4.3 Déroulage de boucle (Loop Unrolling)

Nous avons ajouté un *loop unrolling* pour améliorer l'efficacité du pipeline CPU et réduire l'overhead des boucles.

5 Code Optimisé

Le code optimisé est présenté ci-dessous :

Listing 2: Code optimisé

```
void kernel (unsigned n, float a[n], float b[n], float c[n][n]) {
    unsigned i, j;

    for (i = 0; i < n; i++) {
        float temp = 0.0f;

        // D roulage de boucle (unrolling par 4)
        for (j = 0; j + 4 <= n; j += 4) {
            temp += c[i][j] + c[i][j+1] + c[i][j+2] + c[i][j+3];
        }

        // Gestion des    lments    restants
        for (; j < n; j++) {
            temp += c[i][j];
        }

        a[i] += temp / b[i]; // R duction des divisions
    }
}
```

6 Justification des Optimisations

- **Ordre des boucles** : L'accès mémoire est optimisé pour un parcours *row-major*, améliorant la performance du cache.
- **Loop unrolling** : En déroulant la boucle de 4 itérations, nous réduisons l'overhead de la gestion des boucles et favorisons le parallélisme interne du processeur.
- **Réduction des divisions** : En accumulant les sommes avant la division, nous réduisons le nombre d'opérations coûteuses.

7 Conclusion

Les optimisations apportées permettent de réduire le temps d'exécution du code en améliorant l'utilisation du cache et en minimisant les opérations inutiles. Cette approche garantit une meilleure performance, en particulier pour des valeurs élevées de 'n'.