UNIVERSITÉ CÔTE D'AZUR

MASTER INFORMATIQUE

Rapport de TP

Transformations de boucles

Réalisé par
Francois Flandin

Encadré par
Pr Sid Touati

Master Informatique
à Université Côte d'Azur
Premier semestre de l'annee 2024-2025

Table des matières

1 Introduction

Ce rapport porte sur les transformations de boucles appliquées dans le cadre d'optimisations de code, illustrées par l'exemple de la multiplication de matrices.
L'objectif de ce TP est d'explorer et d'analyser différentes techniques d'optimisation, notamment le déroulage de boucles, la fusion de boucles, la permutation des ordres d'exécution, ainsi que le tuilage ou blocage des boucles.
Ces transformations sont mises en œuvre dans le contexte de performances sur processeurs modernes, où l'exploitation efficace des caches et des pipelines est essentielle.
Les performances ont été mesurées à chaque étape pour évaluer l'impact des techniques proposées.
Ce rapport détaille les approches adoptées, les résultats obtenus, ainsi que les analyses et conclusions tirées de ces expérimentations.

Environnement Expérimental

Micro-Architecture

Dans cette partie sera detaillée la micro-architecture de la machine de tests.

Nom de modèle : 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
Taille des adresses : 39 bits physical, 48 bits virtual
Coeurs physiques : 4
Taille de ligne de cache : 64 octets

Graphical Topology				
Coeurs	0 4	1 5	2 6	3 7
Cache L1	48 kB	48 kB	48 kB	48 kB
Cache L2	1MB	1MB	1MB	1MB
Cache L3	8 MB			

Table 1 – Topologie de la machine de tests

Environnement Logiciel

Dans cette partie sera détaillée la partie logiciel de la machine de test, les fichiers scripts pour changer entre machine allégée et machine classique sont fournis dans l'archive.

Distribution : Fedora Linux v40 WorkStation
Compilateur utilisé : `gcc` version 14.2.1 20240912 avec option `-O2`
Outil de mesure des temps d'exécution : Commande `/bin/time`.

Configuration expérimentale

Processus en activité

Les benchmarks n'ont pas été effectués sur un environnement allégé, pour la curiosité on laissera des fichiers `data_allegee`, avec les résultats des benchmarks réalisés sur une configuration minimale pour chaque exercice.

De part la nature non-allégée de la machine de tests, il ya beaucoup de processus en activité, dont des éditeurs de texte, le terminal, un navigateur et autres applications de messagerie.

Méthodologie de récolte des données expérimentales

Plusieurs scripts ont été réalisés afin de compiler le programme source sous différentes versions pour les besoins du TP, mais aussi afin de réaliser les différents benchmarks.
Pour mesurer le temps d'exécution, on utilisera la commande `/bin/time` sur 4 exécutions consécutives pour en faire la moyenne.
Ces moyennes seront ensuite envoyées a un programme python pour calculer les moyennes de temps d'exécution et leur taux d'accélération par rapport au cas de base, puis en faire un tableau qui puisse être exporté dans un rapport en latex.

2 Déroulage de boucle

Déroulage	Moyenne temps globaux	Taux d'accélération
sans déroulage	**4.624**	**1**
déroulage source	4.326	1.07
funroll-loops	4.390	1.05

Table 2 – Évolution du temps moyen d'exécution en fonction de la méthode de déroulage de boucles.

Les résultats montrent que le déroulage offre bel et bien un gain de performances, jusqu'à 7% pour le déroulage source, de plus, on note que l'option `funroll-loops` offre moins de performances que le déroulage source, mais possède l'avantage que le développeur n'a pas à modifier de lui-même son code.

3 Fusion des boucles

3.1 Analyse résultats

Déroulage	Moyenne temps globaux	Taux d'accélération
sans déroulage	**4.5275**	**1**
déroulage de i	4.5575	0.99
déroulage de i et fusion de j	12.985	0.34

Table 3 – Évolution du temps moyen d'exécution en fonction des déroulages et des fusions des boucles i et j.

Ces résultats nous montrent que le déroulage ou la fusion ne proposent pas de gains de performances sur la machine de test, cependant, la différence est tellement faible, entre le déroulage de i et sans déroulage, que cela peut être dû a un problème venant de la machine de test.

4 Permutations de boucles

4.1 Ordre optimal des boucles

Pour le programme mat_mult.c, l'ordre optimal des boucles (i,j,k) serait (i,k,j).
Cet ordre permet d'optimiser l'accès a la matrice B, car A[i][k] est plus souvent utilisé donc il reste le plus longtemps dans le cache, cependant pour B, parcourir les lignes plutôt que les colonnes entraîne des sauts mémoires, augmentant ainsi le risque de défauts de cache.

4.2 Benchmark des ordres de boucles

Ordre des boucles	Moyenne temps globaux	Taux d'accélération
(i,j,k)	**4.4875**	**1**
(i,k,j)	4.3325	1.03
(j,i,k)	4.4925	0.99
(j,k,i)	4.445	1.01
(k,i,j)	4.46	1.00
(k,j,i)	4.47	1.00

TABLE 4 – Comparaison des différents ordres de boucles sur la moyenne des temps d'exécution.

De ce que l'on peut voir sur ce tableau, l'ordre le plus optimisé est l'ordre (i,k,j) avec un taux d'accélération de 1.03, donc la théorie est valide, de plus, on peut noter que l'ordre le moins performant est (j,i,k).
Cependant, la différence de temps entre les ordres est si fine que cela pourrait être dû a une erreur de mesure ou un autre facteur venant de la machine de test.

5 Tuilage ou blocage de boucles

5.1 Calcul de B, la taille du bloc

La formule pour calculer B est la suivante :

$$3 \times B^2 \times sizeof(float) \leq C$$

`C` dans l'équation représente la taille du cache, en octets, il faut déterminer la taille du cache dans lequel on veut effectuer cette opération.

Cache L1 de la machine test : 48 KiB = $48 * 1024 = 49152$ octets
Cache L2 de la machine test : 1 MiB = $1 * 1048576 = 1048576$ octets
Cache L3 de la machine test : 8 MiB = $8 * 1048576 = 8388608$ octets
Il reste donc a savoir comment trouver B, la taille du bloc, la formule est :

$$[3 \times B^2 \times sizeof(float) \leq C] \Leftrightarrow [3 \times B^2 \times 4 \leq C] \Leftrightarrow [B^2 \times 12 \leq C]$$

$$B^2 \leq \frac{C}{12}$$

Il faut donc l'appliquer pour toutes les tailles de cache.

5.1.1 Taille de B pour le cache L1

$$[B^2 \leq \frac{49152}{12}] \Leftrightarrow [B^2 \leq 4096] \Leftrightarrow [B \leq \sqrt{4096}]$$

$$B \leq 64$$

5.1.2 Taille de B pour le cache L2

$$[B^2 \leq \frac{1048576}{12}] \Leftrightarrow [B^2 \leq 87381] \Leftrightarrow [B \leq \sqrt{87381}]$$

$$B \leq 295$$

5.1.3 Taille de B pour le cache L3

$$[B^2 \leq \frac{8388608}{12}] \Leftrightarrow [B^2 \leq 699050] \Leftrightarrow [B \leq \sqrt{699050}]$$

$$B \leq 836$$

5.1.4 Résultats

On a donc 64 pour la taille du bloc pour le cache L1, 295 pour le cache L2 et 836 pour le cache L3.

5.2 Benchmark de matrix multiply avec différentes tailles de B

Taille de bloc	Moyenne temps globaux	Taux d'accélération
base	**4.4125**	**1**
L1	5.9125	0.74
L2	6.8575	0.64
L3	7.59	0.58

TABLE 5 – Évolution du temps moyen d'exécution en fonction de la taille des blocs.

On note donc que plus on avance dans les caches, plus le temps d'exécution devient long, ce qui est normal, car le temps d'accès aux caches est inversement proportionnel à sa taille. De plus le tuilage donne de pire performances que la version initiale sans tuilage, ce manque de performances pourrait être compensé par une permutation des boucles.

Permutation de boucles source

Dans cette partie sera présente les effets de permutation de boucles pour les différentes tailles de bloc, à travers des tableaux de moyenne de temps et taux d'accélération.

Ordre des boucles	Moyenne temps globaux	Taux d'accélération
(i,j,k)	**5.055**	**1**
(i,k,j)	3.2775	1.54
(j,i,k)	5.0075	1.01
(j,k,i)	5.7525	0.87
(k,i,j)	3.275	1.54
(k,j,i)	5.7775	0.87

TABLE 6 – Évolution du temps moyen d'exécution en fonction de la permutation de boucles sur la taille de bloc L1.

Ordre des boucles	Moyenne temps globaux	Taux d'accélération
(i,j,k)	**6.835**	**1**
(i,k,j)	2.82	2.42
(j,i,k)	6.975	0.97
(j,k,i)	5.9125	1.15
(k,i,j)	2.825	2.41
(k,j,i)	5.86	1.16

TABLE 7 – Évolution du temps moyen d'exécution en fonction de la permutation de boucles sur la taille de bloc L2.

Ordre des boucles	Moyenne temps globaux	Taux d'accélération
(i,j,k)	**7.69**	**1**
(i,j,k)	2.81	2.73
(j,i,k)	7.6775	1.00
(j,k,i)	20.8	0.36
(k,i,j)	2.965	2.59
(k,j,i)	20.33	0.37

TABLE 8 – Évolution du temps moyen d'exécution en fonction de la permutation de boucles sur la taille de bloc L3.

Avec ces benchmarks, on remarque donc que les ordres de boucles les plus optimisés sont (i,j,k) et (k,i,j), ce qui colle avec la théorie détaillée plus tôt, et donne des résultats plutot similaires aux résultats sans le tuilage.
Cependant on remarque que certains ordres offrent de très mauvaises performances, et ce de manière générale, (j,k,i) et (k,j,i).

Tuilage avec -floop-block

Taille de bloc	Moyenne temps globaux	Taux d'accélération
base	**4.66**	**1**
L1	4.92	0.94
L2	6.8575	0.67
L3	7.7075	0.60
floop_block	12.6525	0.36

TABLE 9 – Évolution du temps moyen d'exécution en fonction de la taille des blocs ou de l'option -floop-block.

On constate donc, qu'avec -O2 ainsi qu'avec -floop-block, on observe de bien pires résultats qu'avec le tuilage manuel ou encore une exécution sans tuilage.

6 Conclusion

Au cours de ces expériences, nous avons exploré diverses méthodes d'optimisation centrées sur les boucles. Dans un premier temps, nous avons étudié le déroulage et la fusion de boucles, qui simplifient et réduisent le coût des calculs répétitifs. Ensuite, nous nous sommes intéressés aux permutations de boucles, lesquelles nous ont permis de mieux organiser ces structures en fonction des contraintes imposées par la hiérarchie mémoire.
Enfin, l'étude approfondie du tuilage de boucles nous a offert une compréhension plus fine du fonctionnement des caches et du transit des données à travers les différentes couches de la mémoire. Ces techniques se révèlent essentielles pour maximiser les performances des programmes, en exploitant efficacement les ressources matérielles disponibles.