DM n° 1 — Compression d'image en niveaux de gris

Antoine de Roquemaurel (Groupe 3.1)

1 Compilation exécution et tests

L'archive qui à été déposée sur Moodle est organisé comme suit :

`Report_deRoquemaurelAntoine_G1.1.pdf` Le rapport que vous êtes en train de consulter

`Compressor/` Programme de compression contenant les fichiers détaillés section 1.2.

1.1 Fichier sources

- `block.c` : Fonctions et structures de données concernant les blocks.
- `compressor.c` : Fonctions de compression.
- `decompressor.c` : Fonctions de décompression.
- `main.c` : Fichier principal gérant les arguments
- `ZIterator.c` : Itérateur effectuant un parcours en Z.
- `blockiterator.c` : Itérateur permettant de parcourir une image block par block.
- `dct-idct.c` : Fonctions appliquant la dct et son inverse
- `image.c` : Fonctions et structure de données concernant les images
- `util.c` : Fonctions utiles
- `Makefile` : Fichier Makefile

1.2 Fichiers de tests

L'exécution des tests s'effectue de la même manière que l'archive fournie :

```
aroquemaurel@aokiji < master > : ~/projets/c/oim/jpg-compression/report
[0] % time make tests
dct             [OK]
quantify        [OK]
vectorize       [OK]
compression     [OK]
decompression [OK]
make tests  0,38s user 0,07s system 85% cpu 0,532 total
```

Listing 1 – Execution des tests

Comme montré listing 1, l'intégralité des tests passent, ceux-ci sont basés sur les images d'origine. Un image de taille plus importante à été ajoutée afin de pouvoir mieux observer les différentes optimisation, plus d'informations sont fournies section 4.

2 Compression

Afin d'effectuer la compression, plusieurs tests ont été effectués afin de garantir que chaque module était bien fonctionnel :

– Dct
– Quantify
– Vectorize
– Compression

Chaque test est incrémental : un test ne peut pas passer sans passer le précédent.

2.1 Dct et Quantify

Afin d'appliquer la dct et de quantifier la matrice, il faut utiliser la fonction `applyDct` qui en fonction du 3e paramètre applique soit la normalisation, c'est-à-dire une division par `8.f`, soit il divise par la matrice de quantification. Cette factorisation nous permet de réutiliser cette méthode soit pour le test dct, soit pour le test quantify.

2.2 Vectorize

La vectorisation consiste en l'utilisation d'un Itérateur permettant de parcourir un block en Z comme le montre l'image ci-contre.

Cet itérateur, `ZIterateur`, possède les méthodes `has_next` et `next` permettant de parcourir l'intégralité du block. La méthode `next` doit vérifier tous les cas possibles avant de renvoyer la bonne valeur et de déplacer le curseur dans le tableau.

Une solution plus optimisée que la présente aurait pu être de générer la liste des positions nécessaires à un déplacement en Z avec un script externe tel que Python ou Bash. Cependant cette solution était moins souple en terme d'évolution : l'execution

Une autre solution peut cependant être envisager pour optimiser un peu plus l'algorithme : stocker les valeurs des cases devant être parcourues dans un tableau.

3 Décompression

La décompression utilise globalement les modules déjà développés précédemment, cependant en raison de l'idct particulièrement lente, une étape d'optimisation section 4 sera nécessaire.

Figure 1 – Image mystère

4 Optimisations

Le traitement d'image, et particulièrement la décompression sont des opérations couteuses. Ainsi afin d'avoir un traitement qui soit correct en mémoire ou en cpu, des optimisations étaient nécessaires.

Deux axes d'optimisations ont été travaillés :
– D'une part, limiter les calculs, notamment dans les boucles, et particulièrement pour la fonction idct.
– D'autres part, l'utilisation de plusieurs cœurs en parallélisant le système semblait être intéressant.

Afin de pouvoir observer facilement le temps d'exécution gagné, une image à été ajouté aux scripts de tests. En effet, les images proposées ne dépassaient pas 800×800 pixels, un traitement qui était déjà relativement rapide. J'ai donc téléchargé une image `nasa.pgm` possédant une taille de 1800×1800, cela m'a permis d'effectuer un test de charge tout en pouvant plus facilement observer les différences d'optimisations.

> R Cependant, ces optimisations ne doivent pas altérer la lisibilité du code afin de pouvoir maintenir notre compresseur sans aucun problème.

Section 4.3, figure **??** est disponible une figure montrant les gains de performance obtenus suite aux optimisations.

4.1 Inverse de la dct

4.1.1 Constantes calculées dans les boucles

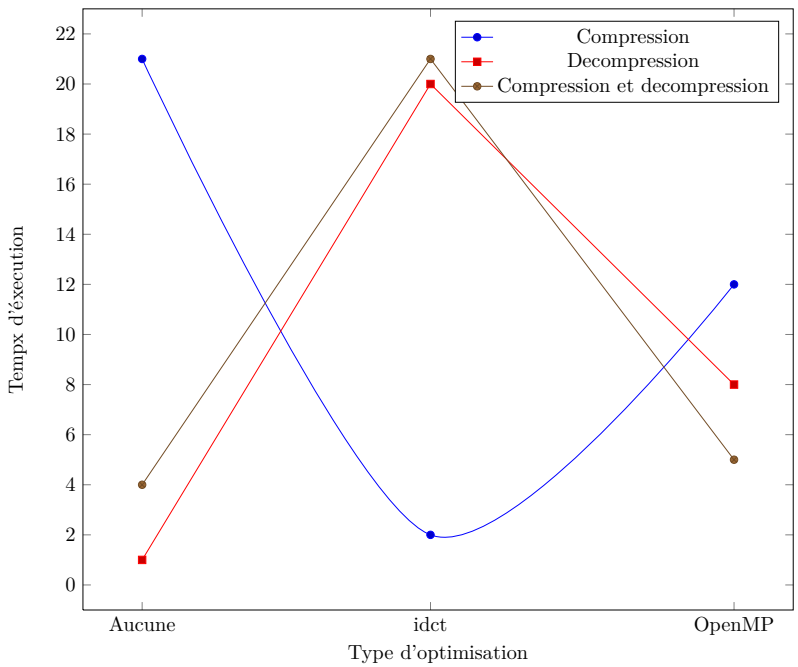
$\frac{\Pi}{16}$
$\frac{1}{\sqrt{2}}$

4.1.2 Cosinus

4.1.3 Déclarations dans les boucles

4.2 Parallélisme avec `openmp`

4.3 Gain de temps grâce à l'optimisation



A Table des figures

1 Image mystère 2

B Listings

1 Execution des tests 1