# HPC - Laboratoire 1

Rayane Annen

4 mars 2024

## Benchmarks

Machine utilisée pour les tests :

- Architecture : AArch64 (ARM)
- CPU : Apple M1 Pro 10 Cores
- RAM : 16 GB LPDDR4X SDRAM

Compilateur :

- clang 15.0.0
- target : `arm64-apple-darwin23.3.0`
- Flags de compilation : `-O3 -g -Wall`
- Librairies : `math.h` et `stb`

### Tests effectués

Images d'entrées :

| Nom du fichier | Dimensions [pixels] | Nombre de composantes par pixel |
|---|---|---|
| `half-life.png` | $2000 \times 2090 = 4038000$ | 3 (8-bit RGB) |
| `medalion.png` | $1267 \times 919 = 1164373$ | 3 (8-bit RGB) |
| `half-life.png` | $1150 \times 710 = 816500$ | 3 (8-bit RGB) |

Nombre de type :

- Tableau 1D : `1`
- Liste chaînée : `2`

Sur la base de ces deux variables (l'image et le type de structures de données) une matrice de tests a été effectuée. Chaque test est effectué 50 fois et le résultat gardé est la moyenne de toutes les runs.

## Résultats obtenus

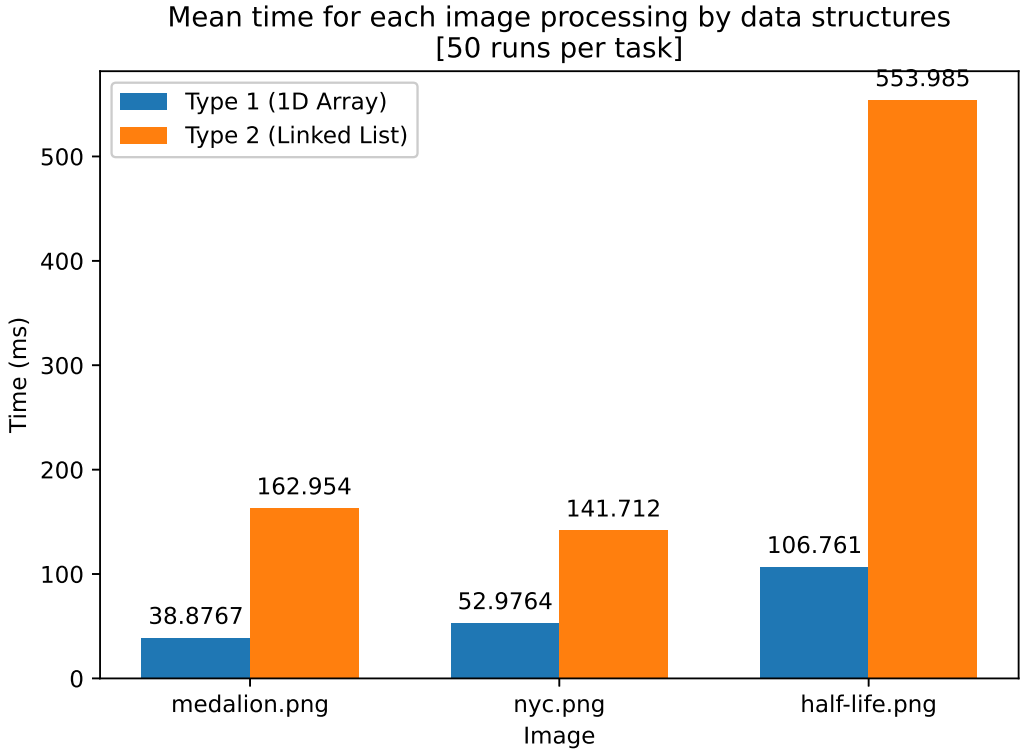


Figure 1: Résultats du benchmark obtenu avec la machine de test.

En temps normal, les résultats en fonction des structures de données devraient être drastiquement différents (un ordre de grandeur x100-x1000 au lieu de x5-x10), en effet, l'accès à un élément dans un tableau se fait en temps constant contrairement à une liste chaînée qui nécessite de re-parcourir toute la liste pour obtenir un élément à un certain indice, ce qui pour une convolution peut être très lent et fastidieux (on doit accéder à 9 éléments de la liste pour chaque pixel, donc potentiellement parcourir la liste jusqu'à atteindre un élément 9 fois)

Ici, les performances pour la liste chaînée sont quand même intéressantes. Afin d'accélérer le calcul, j'ai mis en place un cache le temps du traitement de l'image. C'est un tableau 1D de pointeurs vers les éléments de la liste chaînée, pour le remplir je dois donc parcourir au moins une fois toute la liste.

Cela me permet d'avoir un calcul de convolution quasiment identique pour les listes chaînées et les tableaux unidimensionnels.

Toutefois, le coût de la mise en place du cache impacte directement les performances du traitement (et sur la mémoire), cela reste toujours mieux que de parcourir la liste autant de fois qu'il ne faille accéder à un certain élément.

Un second constat peut être effectué, la taille de l'image détériore les performances (plus elles sont grandes, plus cela prend du temps), c'est normal étant donné qu'on va parcourir pour une étape du traitement au moins $Hauteur \times Largeur \times N_{composantes}$ éléments, ce résultat est multiplié par au moins deux si on est dans la version avec les listes chaînées (on parcourt la liste en intégralité pour construire le cache).

Dans le premier cas on aura de meilleures performances (1D), dans le second (chained) on pourra atteindre les performances optimales au détriment de l'utilisation de la mémoire.