# Travaux Pratique 8

Contexte : Les données indiquées dans ce rapport ont été obtenues avec un GPU Tesla T4 et un processeur Intel Xeon avec 2 cœurs logiques sur Google Colaboratory.

## Exercice 1 :

Ici, chaque thread va s'occuper d'un pixel. On convertit les valeurs rgb ou hsv et on enregistre le résultat dans le(s) tableau(x) de sortie.

Au niveau des temps de calcul, j'obtiens :

| Threads/blocs | Calcul CPU | rgb2hsv | hsv2rgb |
|---|---|---|---|
| 32 * 32 | 10359 | 192 | 165 |
| 1024 * 1 | 11190 | 233 | 173 |
| 1 * 1024 | 10646 | 1273 | 1426 |
| 256 * 4 | 11113 | 206 | 163 |
| 128 * 4 | 10466 | 180 | 151 |
| 128 * 1 | 11782 | 152 | 137 |
| 64 * 1 | 10714 | 148 | 136 |
| 32 * 1 | 10326 | 241 | 233 |
| 32 * 4 | 10533 | 146 | 138 |
| 16 * 4 | 10423 | 144 | 146 |
| 16 * 8 | 10715 | 142 | 149 |

J'ai commencé avec 1024 threads répartis sur deux dimensions. Comme je calcule l'indice du thread avec la formule tid = tid_x + tid_y * width, je me doutais que je devais garder un nombre supérieur pour la dimension en x plutôt qu'en y, afin d'utiliser la coalescence des accès aux données. On le remarque bien sur les temps 1024 * 1 et 1 * 1024. En effet, 1024 * 1 est bien plus rapide. J'ai ensuite approximé le nombre optimal de threads, ainsi que leur répartition. D'après mes résultats, la répartition optimale pour le GPU alloué est de travailler sur 64 ou 128 threads par bloc, répartis de telle façon que la dimension en x est toujours supérieure à la dimension en y, donc de 16 * 4 jusqu'à 128 * 1.

## Exercice 2 :

Ici, on appelle simplement la fonction computeHistogram avec un foncteur qui permet de convertir une donnée en float allant de 0 à 1 exclus, en unsigned allant de 0 à 256 exclus. Je préfère convertir en unsigned plutôt qu'en unsigned char car même si les données vont 0 à 255, c'est plus cohérent avec le type des données en sortie, qui sont en unsigned.

| Images | temps cpu | temps gpu |
|---|---|---|
| Nuit | 1768 | 138 |
| Roy_Lichtenstein_Drowning_Girl | 202 | 143 |
| The_Nightwatch_by_Rembrandt | 2009 | 142 |

Au niveau des temps d'exécution, on a un temps qui semble assez constant niveau gpu, mais peu intéressant pour de petites images.

## Exercice 3 :

On utilise le scan inclusif de OPP::CUDA avec un foncteur qui réalise une addition.

| Images | temps cpu | temps gpu |
|---|---|---|
| Nuit | 0 | 15 |
| Roy_Lichtenstein_Drowning_Girl | 0 | 15 |
| The_Nightwatch_by_Rembrandt | 0 | 15 |

Pour les temps d'exécutions, on a un calcul en temps constant sur cpu et gpu, qui est même plus rapide sur cpu. C'est dû au foncteur qui est une simple addition donc la parallélisation n'est pas très intéressante de manière indépendante.

## Exercice 4 :

Pour cet exercice, j'utilise un kernel qui calcul la valeur transformée $T(xi) = \frac{L-1}{L \times n} r(xi)$ pour chaque pixel.

| Threads/bloc | Hopper cpu | Hopper gpu |
|---|---|---|
| 1024 | 1114 | 52 |
| 512 | 1106 | 46 |
| 256 | 1264 | 44 |
| 128 | 1128 | 43 |
| 64 | 1055 | 45 |

| Threads/bloc | Nuit cpu | Nuit gpu |
|---|---|---|
| 1024 | 2285 | 101 |
| 512 | 2369 | 90 |
| 256 | 2312 | 84 |
| 128 | 2338 | 82 |
| 64 | 2283 | 86 |

| Threads/bloc | Paris cpu | Paris gpu |
|---|---|---|
| 1024 | 701 | 36 |
| 512 | 715 | 32 |
| 256 | 721 | 31 |
| 128 | 737 | 30 |
| 64 | 768 | 32 |

| Threads/bloc | Roy cpu | Roy gpu |
|---|---|---|
| 1024 | 237 | 13 |
| 512 | 238 | 12 |
| 256 | 239 | 11 |
| 128 | 238 | 12 |
| 64 | 270 | 13 |

| Threads/bloc | SunFlowers cpu | SunFlowers gpu |
|---|---|---|
| 1024 | 30224 | 1134 |
| 512 | 29692 | 996 |
| 256 | 30732 | 927 |
| 128 | 30884 | 908 |
| 64 | 29995 | 949 |

Pour chaque image, on constate que le nombre optimal de threads par bloc pour le GPU utilisé est 128. On a une accélération entre qui oscille entre 20 et 30 suivant la taille des données. En effet, on constate aussi que plus l'image est grande, plus la parallélisation est intéressante. (Accélération de 34 pour Sunflowers et 19 pour Roy_Lichtenstein_Drowning_Girl avec 128 threads)