Heterogeneous Programming

Berdoyes Florent et Sembinelli Hugo

Mars 2025

1 Parties du code parallélisées

Nous avons décidé de paralléliser les boucles `for`, en attribuant un thread à chaque pixel.

Nous utilisons des blocs de taille $(16, 16)$, et la grille est composée de $\left(\frac{width+BLOCK\_SIZE-1}{BLOCK\_SIZE}\right)$ blocs en largeur et $\left(\frac{height+BLOCK\_SIZE-1}{BLOCK\_SIZE}\right)$ blocs en hauteur. Ainsi on peut utiliser au maximum 16*16 threads par bloc pour optimiser l'algorithme.

```
// Définition des bloc et grilles
dim3 block(BLOCK_SIZE, BLOCK_SIZE);
dim3 grid((width + BLOCK_SIZE - 1) / BLOCK_SIZE, (height + BLOCK_SIZE - 1) / BLOCK_SIZE);

bilateral_filter_kernel<<<grid, block>>>(d_src, d_dst, width, height, channels, radius, 2 * radius + 1, sigma_color, d_spatial_weights);
```

Figure 1: Code Grid Et Block

2 Comparaison du temps d'exécution entre CPU et CPU+GPU

Nous avons mis un timer au début et à la fin de la fonction main() :

```
cytech@student-laptop:~/Desktop/CUDA/github/heterogeneous_programming/c$ ./exemple lena.bmp test.bmp
Bilateral filtering complete. Output saved as test.bmp
Execution time: 0.4687 seconds
```

Figure 2: CPU Time

Temps d'exécution de exemple.cu (CPU + boucles for parallélisés et calculés avec GPU) :

```
root@2fa30a2f4acd:/workspace/TP3/heterogeneous_programming/Cuda# ./exemple lena.bmp lenaflou.bmp
Bilateral filtering complete. Output saved as lenaflou.bmp
Execution time: 0.3207 seconds
```

Figure 3: GPU Time

3 Image finale

Figure 4: Image filtrée