Dagier Thomas 2ème ISC
Rod Quentin

Rapport sur le TP de Convolution

I . Description et état du projet

Dans le cadre du cours de programmation concurrente, nous devons réaliser une convolution en C sur des images au format ppm. Ce travail doit être réalisé en multi-threading, c'est-à-dire que l'on dédie une partie du travail qui est fait lors de la convolution à un ou plusieurs processeurs.

Au long de ce travail, nous n'avons pas rencontré de difficultés particulières étant donné que nous avions déjà eu à implémenter une convolution dans d'autres cours. Toutefois, la partie sur laquelle nous avons passé le plus de temps a été de faire en sorte que chaque thread aient des tâches équivalentes et ceci, peu importe le nombre de threads donnés en argument à l'exécution.

Avec les commandes "make" puis "./main img_src.ppm, img_dst.ppm 1", nous affichons le temps mis pour réaliser la convolution sur le nombre de threads donné (ici 1).

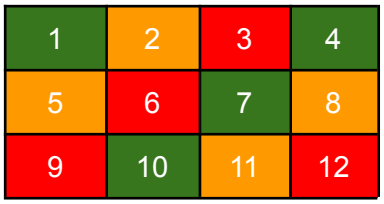
II. Approche concurrente utilisée

Nous avons choisis que la convolution s'effectue par le biais d'une fonction. Dans un premier temps, celle-ci était effectuée en une passe de manière séquentielle. Cette fonction appliquait la convolution un par un sur chaque pixel et retournait la matrice résultante.

Nous avons alors cherché à comment diviser le travail en plusieurs tâches. L'objectif est donc que plusieurs threads soient créés et se répartissent le travail de manière équitable. Nous avons alors transformé notre fonction de convolution pour la rendre concurrente.

Notre solution est la suivante. Chaque thread va traiter plusieurs pixels chacun séparé d'un nombre de pixels constant. Cette distance correspond au nombre de thread - 1. Chaque thread continue à traiter des pixels tant qu'il ne dépasse pas l'image.

Voici un exemple avec trois threads et une image de taille 4x4.



Le thread 1 va traiter tous les pixels de couleur vert, le 2 tous ceux de couleur orange et le 3 ceux de couleur rouge. La différence entre chaque pixel traité par un thread est bien de 2 soit le nombre de threads - 1.

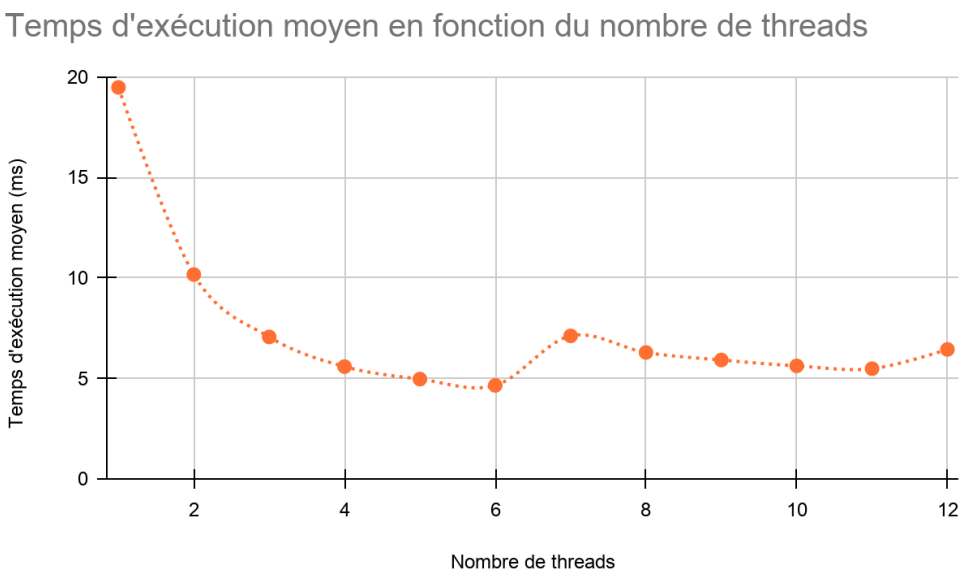
Comme nous pouvons l'observer, les threads ont bien une répartition équitable des tâches. Au maximum, un thread pourrait traiter un pixel de plus qu'un autre.

Chaque thread se base sur la même image source et place le résultat de la convolution du pixel dans la même matrice de destination. Le pixel de destination est écrit une seule fois par un seul thread. Par conséquent, notre matrice de destination ne constitue pas une ressource critique. Il n'y a donc pas nécessité de mettre en place un mécanisme de verrou.

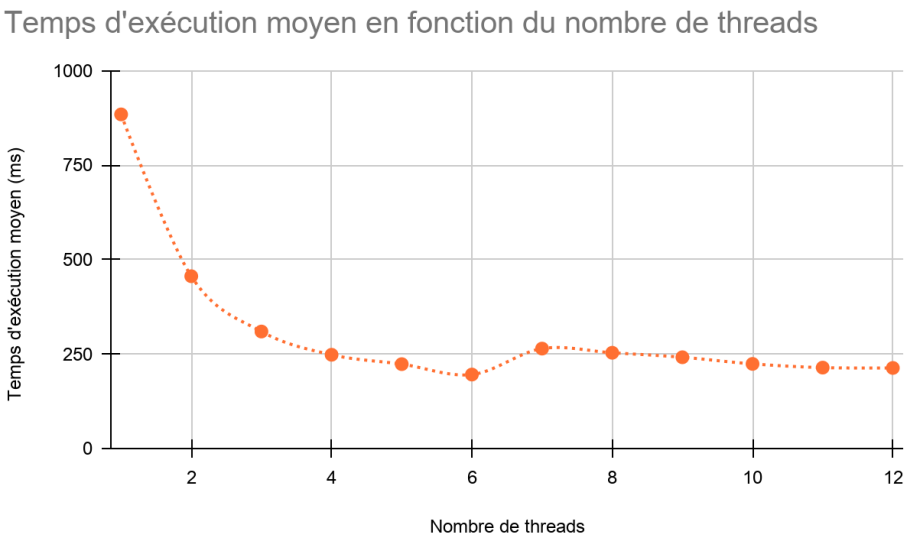
III. Comparaison des performances

Afin de comparer les temps mis pour réaliser la convolution, nous exécutons notre programme pour un nombre de thread allant de 1 à 12 (soit 2 fois le nombre de cœurs présents sur l'ordinateur qui faisait l'exécution). Afin de mesurer un temps d'exécution le plus juste que possible, nous exécutons le programme 5 fois pour chaque incrémentation du nombre de threads. Nous relevons alors la valeur de temps moyenne.

De plus, pour constater des différences entre les exécutions, nous avons réalisé ce processus pour 2 images (fournies dans l'énoncé). La première image fait 7.9 Mb et la seconde 368.4 Mb. Au total, nos résultats sont basés sur 140 exécutions.



Pour la première image, on observe une variation significative du temps. En effet, l'exécution séquentielle (sur 1 thread) dure en moyenne 19.48ms. Assez logiquement, le temps le plus court est celui qui correspond au nombre de cœurs exact de l'ordinateur. Pour 6 threads, on a donc un temps moyen de 4.66ms soit 4,1 fois moins que pour l'exécution séquentielle. On observe aussi que le gain de temps n'est pas du tout linéaire. Plus étrange encore, pour 7 threads, le temps augmente beaucoup (7.12ms) avant de diminuer de nouveau et ceci se confirme également pour la seconde image.



Pour la seconde image, on remarque que la courbe est très similaire à la première. Ce qui nous permet de le dire est la baisse très importante entre les exécutions à 1 thread puis celles à 6 mais aussi et surtout l'augmentation du temps moyen sur les exécutions du 7ème thread avant de recommencer à baisser jusqu'à 12 threads. Au niveau des temps, les exécutions séquentielles ont une durée moyenne de 884.74ms contre 196.35 pour la situation optimale à 6 threads. Le facteur de variation ne change pas beaucoup de la première image puisque sur 6 threads, on met en moyenne 4,5 fois moins de temps que sur 1 thread.

IV . Conclusion

Pour conclure, nous avons séparé un traitement d'image très gourmand en temps et en ressources en plusieurs tâches. Nous avons vu le grand intérêt à exécuter une opération sur plusieurs processus. Cela permet en effet de réduire considérablement le temps d'exécution dans le cas de machine multiprocesseur. Également, créer un nombre de threads dépassant le nombre de processeurs n'est pas une solution optimale. Cependant, la convolution est un cas plutôt simple à rendre multi-tâches. On observe rapidement la complexité de réalisation dans des cas où des sections critiques existent.