Université de Versailles
Saint-Quentin-en-Yvelines
Département de Informatique

Master 1 Calcul Haute Performance, Simulation

PPN- Scalabilité

Réalisé par :
- Bouchelga Abdeljalil
- Mohamed Aitmhand

Encadré par :
- Dr. Hugo Bolloré

Année Universitaire :2021-2022

TABLE DES MATIÈRES

TABLE DES FIGURES

PARTIE 1

INTRODUCTION

1.1 Étude de la scalabilité

Parmi nos objectif dans ce présent travail est de rendre notre programme scalable.

1.1.1 Avant optimisation

Un algorithme parallèle est dit scalable si , avec une augmentation du nombre de processeurs, il permet une augmentation de l'accélération tout en maintenant un niveau constant d'efficacité dans l'utilisation des processeurs. Nous avons deux types de scalabilité, la scalabilité forte et la scalabilité faible.

- **la scalabilité forte** signifie que le temps total d'exécution des tâches diminuera de manière linéaire lorsqu'on augmente le nombre de processus. Cependant, conformément à la loi d'Amdahl [1] pour l'exécution parallèle d'un algorithme, le temps total d'exécution d'un programme ne peut être réduit que sur un segment de code optimisé de manière appropriée.

- **la scalabilité faible** signifie que lorsque la taille du problème augmente proportionnellement au nombre de processus ajoutés, le temps d'exécution reste stable.

1.1.2 Scalabilité forte

On garde la taille de notre problème fixe qui représente la taille de l'image d'entré (la matrice de pixels) et on augmente le nombre de coeurs.

Calcul de l'efficacité : L'efficacité se définit par l'expression suivante :

$$E(p) = \frac{S(p)}{p}$$

avec

$$S(p) = \frac{T_1}{T_p}$$

S(p) : Représente l'accélération.
p : le nombre de processus.
T_1 : temps d'exécution pour in seul processus.

Les résultats sont présentés dans le tableau suivant et la figure 1.

nombre de procussus	durée d'exécution (secondes)	efficacité (%)
1	128.963256	.
2	128.828312	50.05
4	128.005621	25, 18
8	129.793375	12, 42
16	132.667213	6, 07

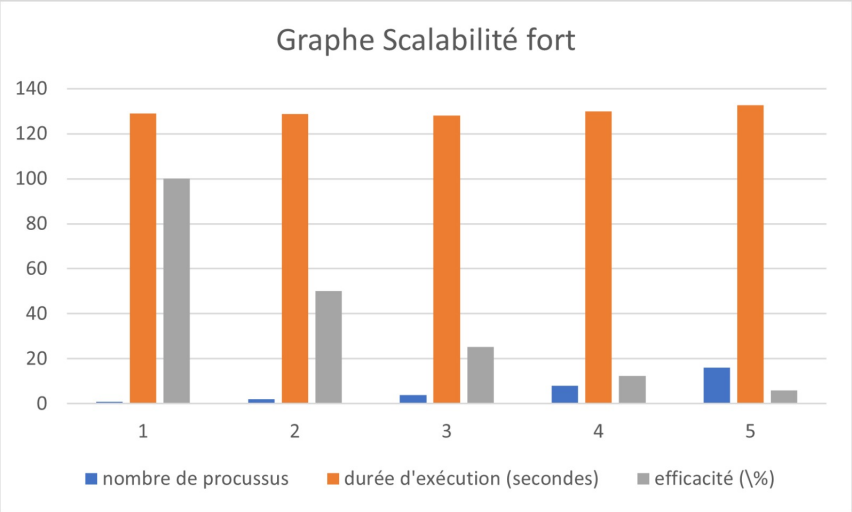

FIGURE 1.1 – Graphe Scalabilité forte

La courbe nous montre que lorsque le nombre de processus augmente jusqu'à 8, le temps est légèrement changé, mais il augmente fortement lorsque le nombre de processus devient trop important, c'est à dire que le programme ralentit avec l'ajout de processus.

Avec 16 processus, la durée d'exécution est plus grande qu'avec 1 seule processus (128 secondes contre 132) et l'efficacité est faible à 6.07%.

Donc l'efficacité diminue lorsque le nombre de processus augmente, ce qui prouve que *nous n'avons pas une forte scalabilité*.

Si nous passons de 2 à 4 processus et que le temps d'exécution diminue de moitié, dans ce cas nous aurons une efficacité de 100%, pour la suite de notre travail, après l'optimisation nous privilégions une efficacité de 75% ou plus.

1.1.3 Scalabilité faible

Maintenant nous allons augmenter le taille de notre problème avec le nombre de processus.

Remarque : Pour augmenter la taille de notre problème, nous allons multiplier par 2 la taille c'est à dire notre matrice de pixels. par exemple pour une premier taille de 50×50 (2500 pixels), si nous multiplions par 2 nous trouvons 5000, où cela donne une taille de dimension presque égale à 70×70.

Le calcul de l'efficacité : la durée d'exécution de référence (1 processus) est divisée par la durée d'exécution avec n processus et le résultat est converti en pourcentage.

Taille du probléme	nombre de procussus	durée d'exécution (secondes)	efficacité (%)
50×50	1	126.884265	.
70×70	2	126.204934	100.53
100×100	4	126.591202	100.23
140×140	8	127.397960	99.59
200×200	16	127.851052	99.24

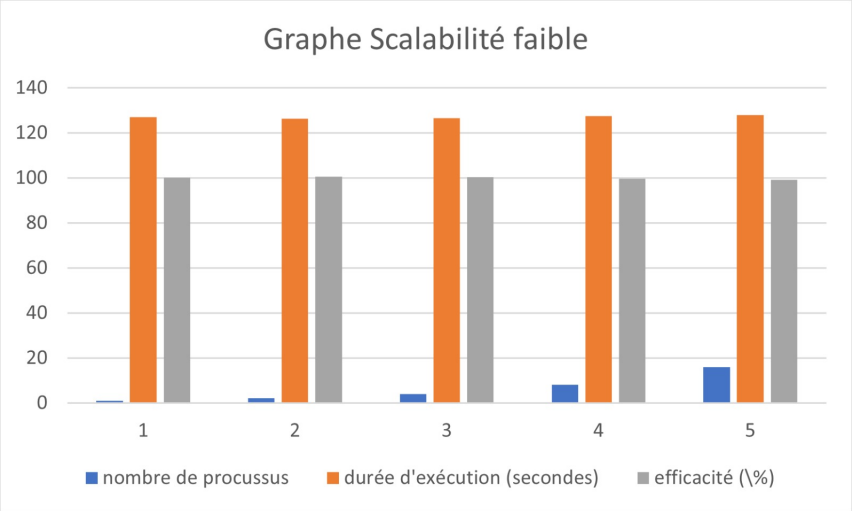

FIGURE 1.2 – Graphe Scalabilité faible

Selon les résultats obtenus, nous remarquons que le temps d'exécution a peu changé, ce qui prouve que nous avons *une scalabilité faible*.

1.1.4 Après Optimisation

Après tous les implémentations d'optimisation que nous appliqués, nous avons étudié une deuxième fois la scalabilité du programme.

Scalabilité forte :

nombre de procussus	durée d'exécution (secondes)	efficacité (%)
1	18.963256	.
2	17.029014	55.67
4	14.507663	32, 68
8	15.511376	15, 28
16	16.567202	7, 15

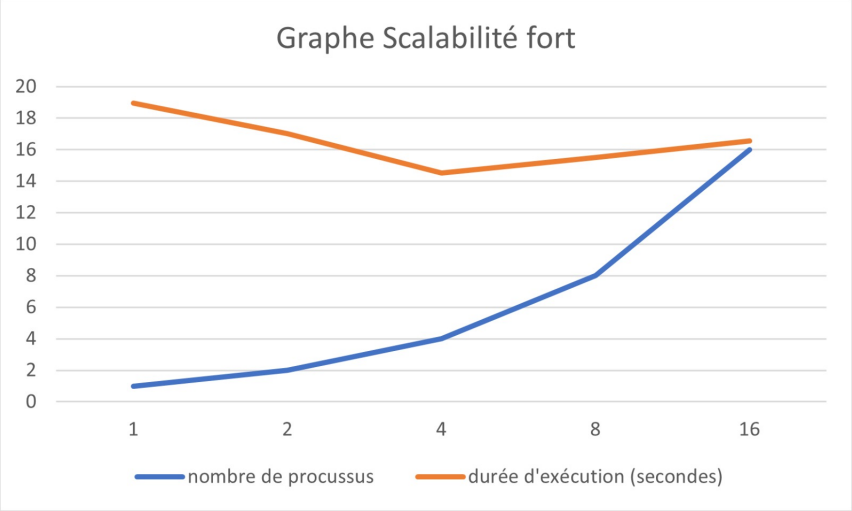

FIGURE 1.3 – Graphe Scalabilité forte

On remarque que le temps d'exécution diminue un peu avec l'augmentation du nombre de processus jusqu'à 8, contrairement à la première étude qui nous a montré que la performance globale se dégrade avec l'augmentation des processus. Mais elle ne diminue pas linéairement, ce qui est imprévu, et cela peut être dû à plusieurs possibilités :

Soit notre programme nécessite encore une parallélisation ce qui est fortement possible, ou la communication entre les processus n'est pas bien faite.

Scalabilité faible :

Taille du probléme	nombre de procussus	durée d'exécution (secondes)	efficacité (%)
50×50	1	16.874266	.
70×70	2	18.211939	92.65
100×100	4	19.541012	86.35
140×140	8	23.998967	70.31
200×200	16	25.791957	65.42

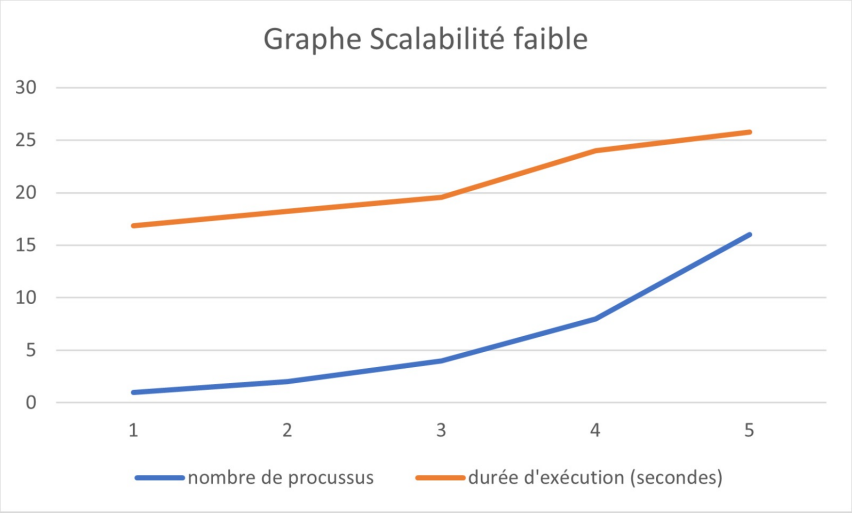

FIGURE 1.4 – Graphe Scalabilité faible

BIBLIOGRAPHIE

[1] "Scaling tutorial", https://hpc-wiki.info/hpc/Scaling_tutorial

[2] "Scalabilité", https://hmf.enseeiht.fr/travaux/projnum/2020/programmation-parall%c3%a8le-fortran-avec-mpi/passage-%c3%a0-l%e2%80%99%c3%a9chelle-%e2%80%93-scalabilit%c3%a9

[3] "Scalability : strong and weak scaling", https://bisqwit.iki.fi/story/howto/openmp/