Durant notre collecte de données, nous avons pu constater que l’approche de regroupement de threads était celle la plus efficace pour le traitement de données que nous avions à faire que l’approche séquentiel et parallèle concurrente.

En premier lieu, lorsque nous comparons les données de vitesse séquentielle avec ceux des deux autres on constate rapidement que cette approche ne devient plus viable dès que l’on nécessite une performance accrue. En effet, lorsque nous n’utilisons qu’un seul cœur pour effectuer un grand nombre de transformations, une grande partie des ressources disponibles de l’ordinateur sont simplement en attente et donc ne contribuent pas à augmenter la vitesse d’exécution des transformations. Ainsi, plus on augmente le nombre de fichier plus l’écart entre les solutions utilisant de la multiprogrammation ce fait ressentir. Cette approche n’est cependant pas à rejeter dans toutes les situations, car dans un cas où il serait possible de prendre plus de temps à effectuer une tâche sans altérer l’expérience de l’utilisateur, il serait judicieux de l’envisager puisque celle-ci ne prend pas beaucoup de ressource du processeur lui permettant ainsi de se concentrer sur d’autre tâche et de ne pas le surcharger.

En deuxième lieu, en analysant une fois encore les données de notre tableau, on peut voir que même si l’approche parallèle concurrente semble être pratiquement similaire en termes de vitesse dans ses débuts, elle devient moins performante au fur et à mesure que nous augmentons la taille de fichier à traiter. Puisque les threads sont lancés sans régulation lorsque la charge de traitement augmente les threads doivent alors se disputer les ressources ce qui ralentit le processus principale. Le processeur doit prendre une partie de ses ressources pour empêcher qu’il n’y ait un conflit d’accès dans la mémoire, ainsi si les fichiers sont de taille supérieur le temps auquel chaque thread doit avoir accès à l’espace mémoire est nécessairement augmenter par la quantité de ligne à traiter , ce qui ralentit les threads qui doivent attendre que les ressources soient libéré.

En conclusion, le regroupement de threads est donc la solution la plus efficace dans notre situation, l’approche séquentielle ne pouvant supporter une grande quantité de fichier et celle parallèle concurrente forçant le processeur à agir comme coordonnateur des ressources. De plus, étant donné que le regroupement de threads gère lui-même les ressources du processeur il est possible d’utiliser l’entièreté de sa puissance et donc de d’augmenter la vitesse de traitement.