# Rapport Labo 2 – Semaine 2

## Conditions de l'expérience

Le but de cette expérience est de nous faire explorer différentes stratégies nous permettant d’réaliser des lectures-écritures dans des ficher, tout en nous faisant porter une attention particulière à leur efficacité. Pour ce faire, nous allons réaliser les étapes suivantes :

1. Comprendre le fonctionnement du programme mettant en œuvre les entrées sorties avec tampon.
2. Modifier le programme pour récupérer les données qu’il génère au format CSV.
3. Analyser les données retournées par le programme pour enfin pouvoir discuter de l’efficacité des différentes stratégies de lecture-écriture utilisées.

## Présentation des mesures

### Tableau des mesures



Il est important de noter qu’à chaque fois la même séquence de bytes, soit 10485760 bytes, est écrite dans un fichier de manière à pouvoir interpréter correctement les résultats. Seul la stratégie d’écriture change, de cette manière, il sera possible de comparer de manière correcte le temps d’exécution de chaque stratégie.

### Graphiques :

Figure 1 Figure 2

Figure 3 Figure 4

L’axe des abscisses représente la taille des bloques en bytes.

L’axe des ordonnées représente le temps en millisecondes.

Une taille de block de 0 représente une écriture bytes par bytes.

## Analyse de ces mesures

### Write

Sur le graphique de la figure 1, il est visible que, plus la taille de block à bufferiser est grande, plus l’écriture dans un fichier est rapide. Il en va de même pour le graphique de la figure 2 où cette fois-ci, les bloques sont écrits bytes par bytes dans un fichier. En comparant ces deux graphiques, il est tout de suite visible, que les opérations d’écritures avec l’utilisation d’un buffer sont plus efficaces que celles réalisées sans buffer.

### Read

Sur le graphique de la figure 3, il est visible que, plus la taille de block à bufferiser est grande, plus les opérations de lecture dans un fichier sont rapide. Il en va de même pour le graphique de la figure 4 où cette fois-ci, les bloques sont lu bytes par bytes. En comparant ces deux graphiques, il est tout de suite visible que les opérations de lecture avec l’utilisation d’un buffer sont plus efficaces que celles réalisées sans buffer.

D’une manière générale, la lecture ou l’écriture par block de donnée bufférisée est plus efficace que la seconde stratégie sans buffer. En effet, elle permet de limiter le nombre d’appels systèmes nécessaire à la lecture ou l’écriture des données. Tout en sachant que les appelles systèmes sont couteux en terme de temps d’exécution.

## Modifications apportées au code source

Pour récupérer le résultat de l’exécution du programme au format « csv », j’ai simplement ajouter un conteneur de type « ArrayList » ainsi qu’un champ de type string aux champs privés de la classe « BufferedIOBenchmark ». Le champ de type string stocke temporairement les données qui nous intéresse. Ces dernières sont récupérées dans les méthodes « produceTestData » et « consumeTestData ». A chaque fois que ces méthodes sont appelées dans le main, les données utiles sont récupérées et formatées dans la string au format csv puis enregistrées dans le conteneur. Finalement j’ai créer une méthode « toCsvData » qui permet d’écrit, de manière optimisée, les données récoltées par le conteneur, dans un fichier. Pour plus de détails, je vous invite à regarder le code source disponible sur le lien GitHub.