Mini-projet système d'exploitation

Couedor Léo

UBO
Année Scolaire 2021-2022
Système d'exploitation

1 – Description du programme C

Le programme créé permet d'additionner deux à deux les éléments de deux tableaux, ceci avec l'utilisation des threads. Le programme est subdivisé en trois fonctions, et un main permettant son lancement.

La première fonction, nommée 'generateArray()' permet de créer un tableau d'une certaine taille, définie selon la constante NBELEM, et d'initialiser les valeurs de ce tableau à des valeurs aléatoires.

Une seconde fonction, 'additionTab()', reçoit en paramètre les pointeurs sur les deux tableaux dont les valeurs doivent être additionnées, un chiffre désignant la méthode à utiliser, un numéro désignant le nombre de threads qui doivent être créés, et un chiffre indiquant s'il doit ou non y avoir migration de thread entre cœurs. Cette fonction crée dans un premier temps un tableau destiné à contenir tous les threads, ainsi qu'un second tableau résultat de l'addition des valeurs des deux premiers. La fonction crée ensuite le nombre de threads souhaité, en leur attribuant la fonction 'add()' et en lui passant en paramètre une structure de données contenant les adresses des deux tableaux à additionner, l'adresse du tableau résultat, le numéro du thread créé, la méthode à utiliser, ainsi que le nombre de threads créés.

Cette fonction 'add()' permet d'effectuer l'opération d'addition en fonction de la méthode, avec un parcours des tableaux différents selon les trois cas possibles : la répartition cyclique des éléments, la répartition cyclique des blocs d'éléments, et la répartition à la demande des éléments.

Enfin, la fonction 'main()', elle reçoit en paramètre les informations de méthode, nombre de threads, et migration, qui seront ensuite à fournir à la fonction 'additionTab()'. La fonction lance ensuite l'addition de deux nouveaux tableaux à additionner, et ce huit fois consécutives, le tout en chronométrant la durée d'exécution, permettant ensuite de calculer le temps moyen d'exécution pour une configuration. Enfin, le programme écrit ensuite ces données à la suite d'un fichier CSV, précédemment créé dans le cas où il n'existe pas.

2 – Description du script Shell

Le script Shell commence par tenter la suppression du fichier CSV, ceci pour ne pas mélanger les données de deux exécutions, le mode d'insertion du programme C étant en ajout, et non seulement en écriture. Il compile ensuite le programme, pour s'assurer de lancer la dernière version du programme.

Le programme nécessite trois paramètres pour être lancé. Le script est donc constitué d'une imbrication de trois boucles permettant de tester l'ensemble des valeurs. La première boucle teste les valeurs de 1 à 3, pour tester sur les trois méthodes possibles. La seconde boucle teste sur des valeurs numériques jusqu'à 10, pour 2^10 = 1024, le nombre maximal de threads. Enfin, la dernière boucle teste les valeurs 0 et 1 pour indiquer s'il doit y avoir migration ou non. Enfin, le script lance le programme précédemment compilé avec les valeurs de ces trois boucles en paramètre, pour tester toutes les combinaisons possibles.

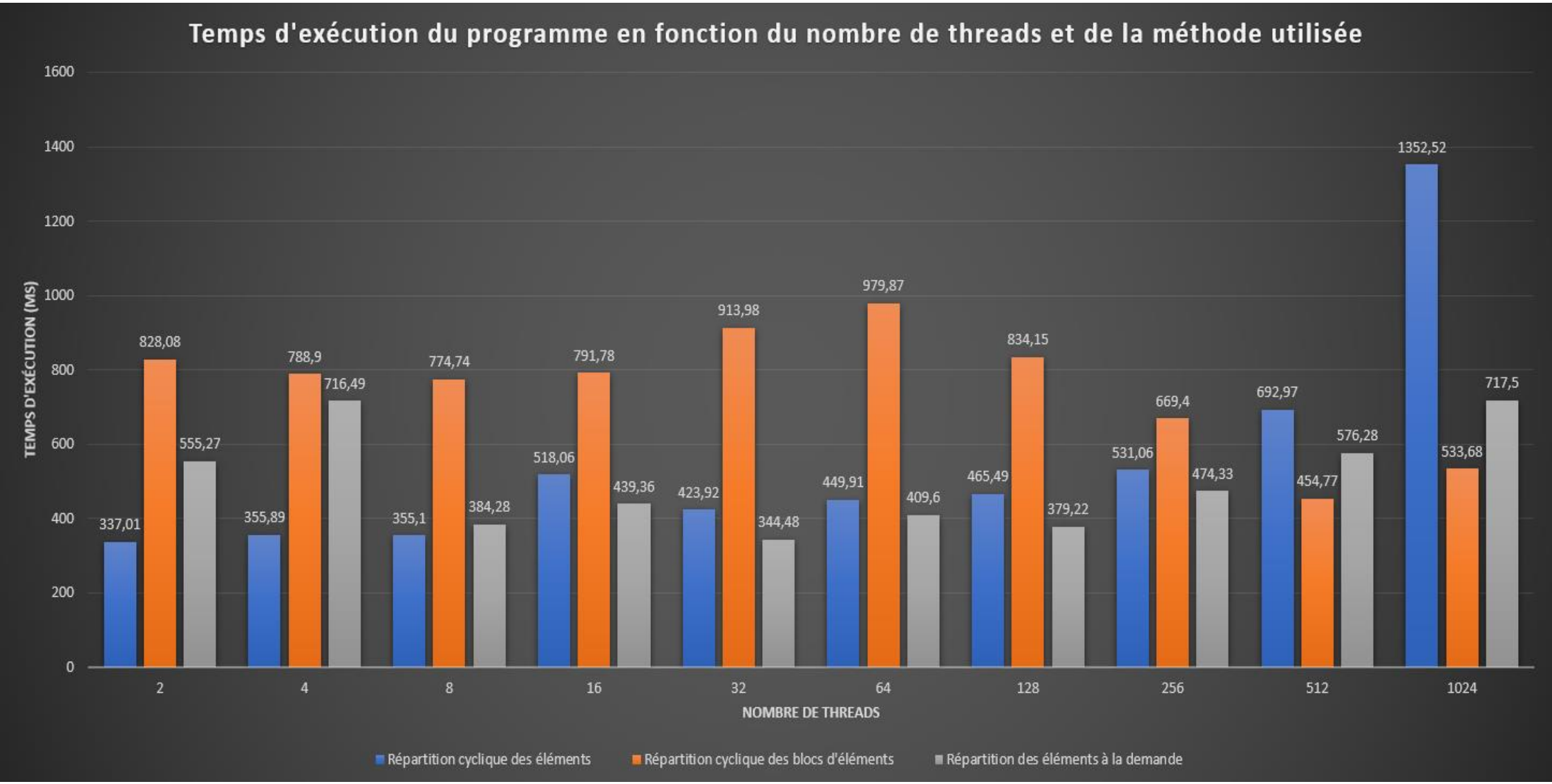
3 – Analyse des données

Les données ayant été écrites dans un fichier CSV, après quelques manipulations, elles ont ensuite pu être représentées sous forme de graphique pour déterminer la meilleure configuration possible. Ces données ont été obtenues pour la configuration suivante :

- Tableaux de 10 000 éléments
- Blocs de 200 éléments
- 8 mesures pour obtenir les temps d'exécution moyens
- 1024 threads au maximum
- Machine virtuelle 5 cœurs

Vous trouverez à la page suivante les données obtenues , sous forme de graphique.

La migration des threads entre les cœurs n'étant pas fonctionnelle, les deux seuls paramètres influant sur la durée d'exécution du programme sont donc le nombre de threads ainsi que la méthode utilisée, qui sont donc les deux seules données représentées sur le graphique.


Temps d'exécution du programme en fonction du nombre de threads et de la méthode utilisée
TEMPS D'EXÉCUTION (MS)
1600
1400
1200
1000
800
600
400
200
0
337,01
828,08
555,27
355,89
788,9
716,49
355,1
774,74
384,28
518,06
791,78
439,36
423,92
913,98
344,48
449,91
979,87
409,6
465,49
834,15
379,22
531,06
669,4
474,33
692,97
454,77
576,28
1352,52
533,68
717,5
2
4
8
16
32
64
128
256
512
1024
NOMBRE DE THREADS
Répartition cyclique des éléments
Répartition cyclique des blocs d'éléments
Répartition des éléments à la demande

Pour cette configuration, plus particulièrement pour cette quantité de données, nous pouvons observer que la meilleure solution est de minimiser le nombre de threads et d'utiliser la méthode de répartition cyclique des éléments. Cette méthode semble par contre devenir extrêmement peu efficace au fur et à mesure que le nombre de threads augmente, jusqu'à devenir la solution la moins efficace à partir de 512 threads.

La répartition à la demande des éléments semble quant à elle plus constante, moins dépendante du nombre de threads, et semble trouver son utilité sur un nombre intermédiaire de threads.

La répartition cyclique des blocs d'éléments devient de plus en plus efficace au fur à et mesure que le nombre de threads augmente, tandis que l'inverse se produit pour les deux autres solutions, ce qui en fait de loin la solution la plus efficace pour un grand nombre de threads. Nous pouvons d'ailleurs conjecturer que cette solution serait plus efficace encore pour un grand nombre de threads et un plus grand nombre de données, comparé aux deux autres méthodes.

Les trois solutions ont donc toutes leurs qualités et leurs défauts, et ne semblent donc pas être destinées aux mêmes usages. La méthode de répartition cyclique des éléments semble d'avantage destinée à un faible nombre de threads. La méthode de répartition à la demande semble plus adaptés pour des grandes quantités de données avec un nombre important de threads. Enfin, la méthode de répartition cyclique des blocs d'éléments semble plus polyvalente, ce qui en ferait un bon choix pour un programme pour lequel le nombre de threads et de données n'est pas renseigné.