UNIVERSITÉ
LAVAL

TP 1 - PThread

Réalisé par **l'équipe 1** constitué de :

Rémi Huguet - 536 795 651
Gabriel Phillipon - 536 795 815

Dans le cadre du cours :
GIF-7104 – Programmation parallèle et distribuée

Travail présenté à :
Marc Parizeau

Remis le :
10 Février 2021

Objectif

L'objectif de ce TP est de trouver tous les nombres premiers dans une série d'intervalles, grâce à un programme multifilaire utilisant la librairie PThread. Le but est également de mesurer le Speedup entre l'implémentation séquentielle et l'implémentation parallèle de ce programme.

Solution

Solution - Diviser le nombre d'intervalles

Dans un premier temps le but a été de repartir le nombre d'intervalles des fichiers à travers plusieurs threads. En effet, de très gros fichier (comme le fichier `8_test.txt`) contiennent un grand nombre d'intervalles qu'il faut repartir en sous-groupe d'intervalles. Ainsi dans un premier temps on construit T (nombre de threads) sous-groupes d'intervalles contenant un nombre égal (ou presque, tout dépend du reste) d'intervalles à traiter.

Ensuite, on sépare également notre structure de données en tableaux de T éléments. Enfin chaque threads va gérer un sous-groupe d'intervalles. À la fin on joint les résultats des tableaux, pour récupérer tous les nombres premiers.

> Prenons par exemple les intervalles suivants : [5,20], [22,34], [45,60], [63,100], [102,120], [126,145], [152,185], [187,200]
> Pour un nombre de thread égal à 4, ils vont être gérés comme suit :
>
> — Le thread 1 va gérer : [5,20], [22,34]
>
> — Le thread 2 va gérer : [45,60], [63,100]
>
> — Le thread 3 va gérer : [102,120], [126,145]
>
> — Le thread 4 va gérer : [152,185], [187,200]

Cette solution permet d'éviter les Mutex. En effet, on écrit les nombres premiers trouvés dans des vecteurs différents . Il n'y a donc pas besoin de Lock & Unlock le vecteur de résultat qui contient les nombres premiers. Les threads pour cette solution sont donc complètement indépendants. De plus, comme les intervalles sont trié, et que chaque thread s'occupe séquentiellement d'un ensemble d'intervalles, les nombres premiers sont écrit dans l'ordre.

Spécification Matériel

— OS : Ubuntu 20.10

— Processeur : Intel® Core™ i7-6700HQ CPU @ 2.60GHz × 8

— RAM : 8 Go

Résultat

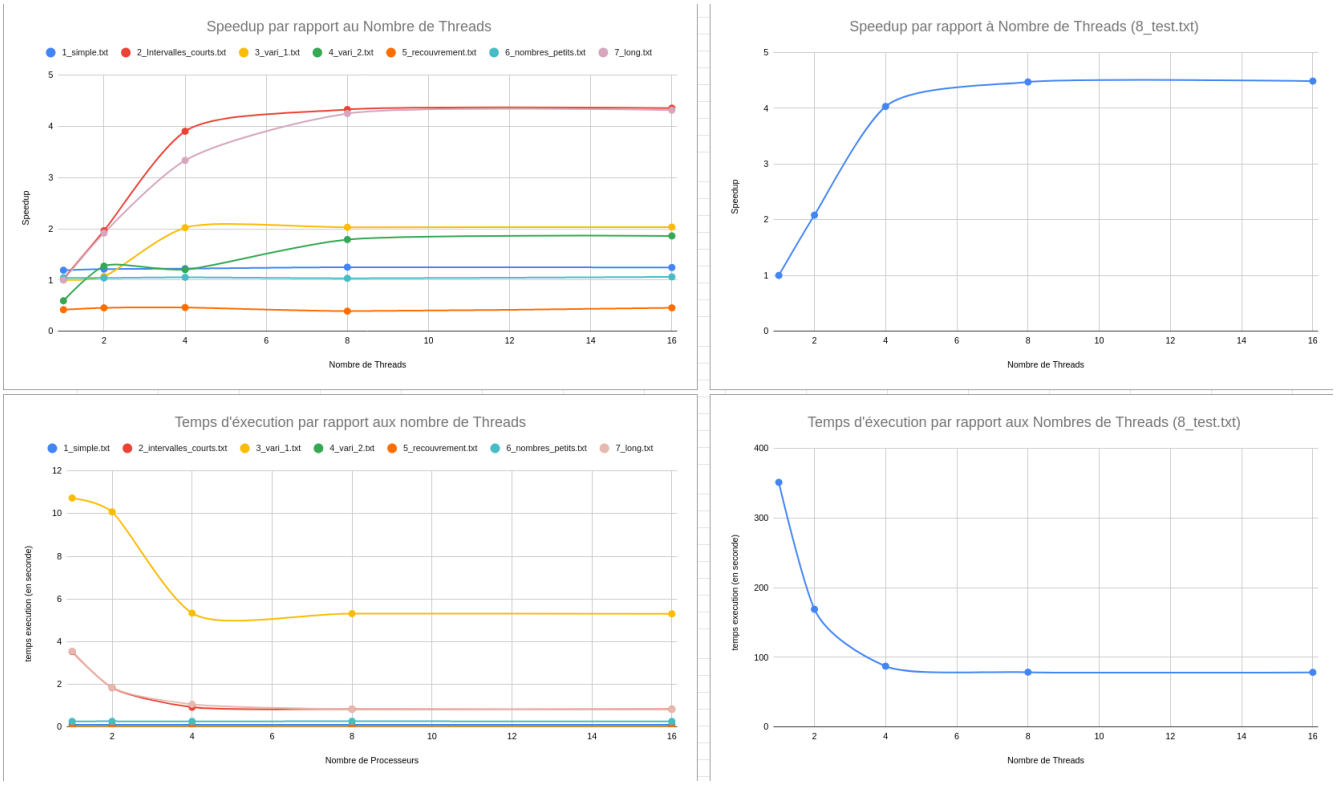

FIGURE 1 – Speedup et temps d'exécutions - Solution 1

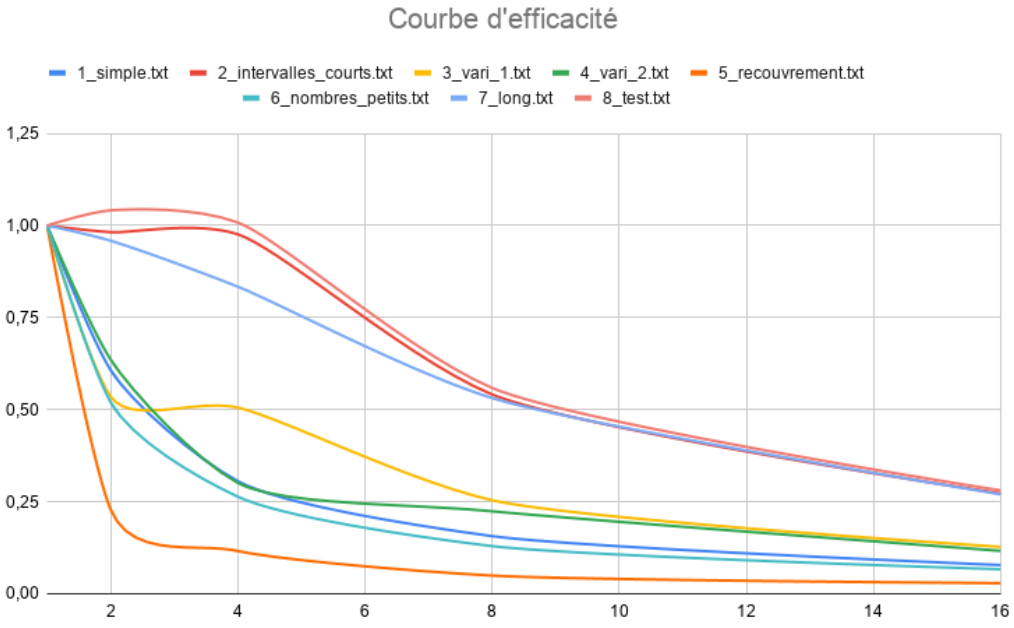

FIGURE 2 – Courbe d'efficacité pour les 8 fichiers

Analyse

Notre solution est conforme à ce que l'on attendait. Elle est efficace sur les fichiers comportant un grand nombre d'intervalles comme le montre le tableau ci-dessous : Plus le nombre d'intervalles est grand, plus le Speedup est important (et inversement).

Fichier	8_test	7_long	2_intervalle_courts	4_vari_2	3_vari_1
Nombres Intervalles	100028	993	1025	9	6
Speedup (8 threads)	Bon (4,5)	Bon (4.25)	Bon (4.3)	Moyen (1.9)	Moyen (2.1)

Fichier	5_recouvrements	1_simple	6_nombres_petits
Nombres Intervalles	2	1	1
Speedup (8 threads)	Mauvais (0.4)	Mauvais (1.2)	Mauvais (1)

De plus, on remarque qu'à partir de 8 threads, le speedup stagne. En effet, c'est le comportement normal, puisque sur la machine de test nous avons un processeur 8 core ; donc à partir de plus de 8 threads, le programme commence à travailler en concurrence et non plus en parallèle et donc ralentit.

Optimisation

Pour tirer un maximum partie de la machine sur lequel le programme a été exécuter et afin d'optimiser les temps d'exécution, la compilation est faite en `"-O3"` (voir `CMakeLists.txt`). Pour ce qui est des structures de données, nous avons préféré les types `Tuples` (std) au `struct`, puisqu'elles sont légèrement plus rapide. Par ailleurs, étant donné qu'aucun nombre premier n'est pair (hormis 2), on ne traite que les nombres impairs.

Piste d'amélioration

Notre solution ne convient pas dans le cas ou le nombre d'intervalles est inférieur ou égal au nombre de threads demandés.

La solution est donc de diviser les intervalles de façon à avoir au moins un intervalle par threads. En effet, avec cette solution, le speedup est bien meilleur pour les fichiers `1_simple.txt` et `6_nombres_petits` comme on peut le voir sur le tableau ci-dessous.

Fichier	1_simple	6_nombres_petits	5_recouvrements
Nombres Intervalles	1	1	2
Speedup (8 threads)	Bon (3.27)	Moyen (1.6)	Mauvais (0.2)

Par ailleurs, une autre piste d'amélioration serait dans le cas du recouvrement. En effet, on se retrouve avec des intervalles dont le nombre de nombres premiers à vérifier est beaucoup plus grands pour certains threads que d'autres. Ainsi, il serait pertinent d'essayer de minimiser l'écart de quantité de traitement entre les différents threads afin d'éviter que certains soient inactifs.