Thi Thu Uyen Nguyen & Hiba Hammi

SR02 - Rapport TD08

1. Tâche 1

Après initialisation d'un tableau indexé de 2 à 20 à vrai, pour i de 2 à 4 ($\lfloor\sqrt{20}\rfloor$) on a :

- $i = 2, a[i] = vrai$

  Pour $j = 4, 6, 8, 10, 12, 14, 16, 18, 20$, mettre à jour $a[j] = faux$
- $i = 3, a[i] = vrai$

  Pour $j = 9, 12, 15, 18$, mettre à jour $a[j] = faux$
- $i = 4, a[i] = faux$

Résultat : les nombres premiers de 1 à 20 sont : 2, 3, 5, 7, 11, 13, 17, 19.

La deuxième boucle commence à i^2 car les nombres divisibles par i entre i et i^2 sont déjà traités par les boucles précédentes, puisqu'au moment où l'on arrive à la $i-$èmè boucle, toutes les valeurs inférieures à i ont déjà été évaluées, et $a[i*x]$ auront déjà été mis à faux, quelque soit $x < i$. De plus, commencer à i ou 0 induirait une mise à faux de l'index i, alors que ce dernier, si il est évalué, est un nombre premier.

La première boucle s'exécute jusqu'à $\sqrt{n}$ car au dessus de $\sqrt{n}$, la boucle de l'intérieure commencerait à une valeur supérieure à n^2, et serait donc immédiatement arrêtée (car la boucle intérieure est entre i^2 et n). Si $\sqrt{n}$ n'est pas un entier, on prendra la valeur entière inférieure, puisque la valeur entière supérieure conduirait à initialiser la boucle intérieure avec une valeur supérieure à n.

2. Tâche 2

Pour compiler le programme, il suffit de lancer:

```
gcc tache2.c -o tache2
-lm
```

Voici le résultat obtenu pour N = 120

```
hammihib@DESKTOP-5EJOHB1:/mnt/c/Users/zerat/Desktop/Hiba/SR02/TD 8 threads$ ./tache2
Entrez la valeur N:
120
Il y a 30 nombres premiers <= 120 :
Temps d'execution : 0.000086 secondes
```

3. Tâche 3

Les tests sont réalisés depuis la machine dont les caractéristiques sont les suivantes:

```
hammihib@DESKTOP-5EJOHB1:/mnt/c/Users/zerat/Desktop/Hiba/SR02/TD 8 threads$ sudo lshw -short
H/W path  Device  Class      Description
========================================
                  system     Computer
/0                bus        Motherboard
/0/0              memory     15GiB System memory
/0/1              processor  Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
/1        wifi0   network    Ethernet interface
/2        wifi1   network    Ethernet interface
/3        wifi2   network    Ethernet interface
```

Pour compiler le programme, il suffit de lancer:

```
gcc tache3.c -o tache3 -lm
-lpthread
```

En testant le programme tache3.c, voici les résultats obtenus pour $N = 120$ et avec $1, 2, 3, 4, 7$ threads :

```
uyennguyen@DESKTOP-K8DNSOT:~/SR02/SR02-OperatingSystemConcepts/Threads$ ./tache3
Entrez un nombre N:
120
Entrez le nombre de threads:
1
Il y a 30 nombres premiers <= 120 :
Temps d'execution : 0.000698 secondes
```

```
uyennguyen@DESKTOP-K8DNSOT:~/SR02/SR02-OperatingSystemConcepts/Threads$ ./tache3
Entrez un nombre N:
120
Entrez le nombre de threads:
2
Il y a 30 nombres premiers <= 120 :
Temps d'execution : 0.002025 secondes
```

```
uyennguyen@DESKTOP-K8DNSOT:~/SR02/SR02-OperatingSystemConcepts/Threads$ ./tache3
Entrez un nombre N:
120
Entrez le nombre de threads:
3
Il y a 30 nombres premiers <= 120 :
Temps d'execution : 0.005542 secondes
```

```
uyennguyen@DESKTOP-K8DNSOT:~/SR02/SR02-OperatingSystemConcepts/Threads$ ./tache3
Entrez un nombre N:
120
Entrez le nombre de threads:
4
Il y a 30 nombres premiers <= 120 :
Temps d'execution : 0.001893 secondes
```

```
uyennguyen@DESKTOP-K8DNSOT:~/SR02/SR02-OperatingSystemConcepts/Threads$ ./tache3
Entrez un nombre N:
120
Entrez le nombre de threads:
7
Il y a 30 nombres premiers <= 120 :
Temps d'execution : 0.001941 secondes
```

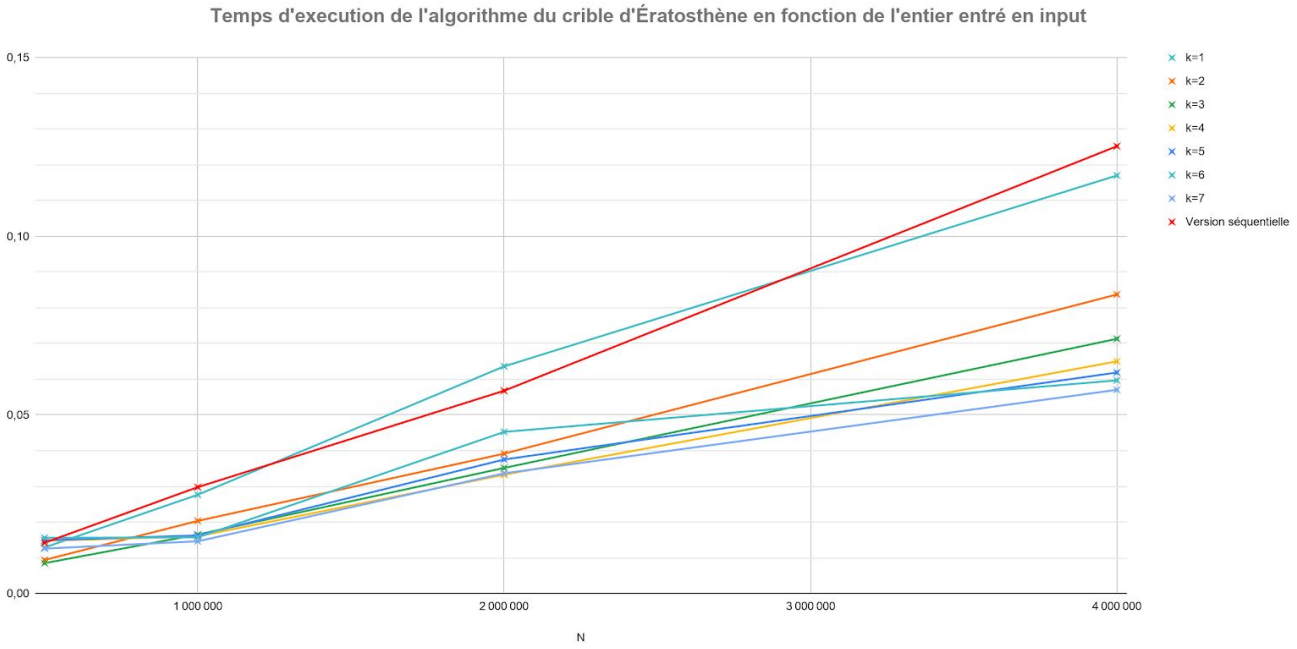
4. Tâche 4

Afin de réaliser les mesures, nous avons lancé le programme pour les différentes valeurs indiquées dans l'énoncé. A chaque itération, nous avons récoltés 50 mesures successives, puis avons effectué une moyenne dont les résultats sont disponibles dans le tableau suivant:

N	500 000	1 000 000	2 000 000	4 000 000
k=1	0,01289388	0,027655	0,06359701	0,11702414
k=2	0,00940617	0,02034113	0,03916162	0,08371081
k=3	0,00850844	0,01652366	0,03516329	0,07129905
k=4	0,01475195	0,01603023	0,03326214	0,06499469
k=5	0,01490869	0,01628062	0,03752821	0,06186791

k=6	0,01561869	0,01573743	0,04524769	0,05968006
k=7	0,01258478	0,01463363	0,03365428	0,05702351
Version séquentielle	0,0142135	0,02979126	0,05677259	0,12521726

Titre: Tableau des moyennes des résultats des temps d'exécution en secondes de l'algorithme du crible d'Ératosthène en fonction du nombre donné en entrée N



Intervalle de confiance pour $N = 500\,000$. Soit un niveau de risque alpha de 5%

Moyenne : M
Coefficient critique : **t = 1.960** pour alpha 5%
Ecart Type : E
Taille échantillon : N
Intervalle de confiance : I

$$I = [\, M - t * E / racine(N) \;;\; M + t * E / racine(N) \,]$$

	IC inf	**N = 500000**	**IC sup**
k=1	0,012506877 01	0,012893 88	0,013280882 99
k=2	0,009275470 246	0,009406 17	0,009536869 754

k=3	0,008407321504	0,00850844	0,008609558496
k=4	0,014206449311	0,01475195	0,015297450609
k=5	0,014020323552	0,01490869	0,015797056648
k=6	0,0150865702	0,01561869	0,0161508098
k=7	0,0119837173	0,01258478	0,0131858427
Version séquentielle	0,013759538003	0,0142135	0,014667461977

Interprétation les résultats obtenus : Lorsque le nombre de threads augmentent, les temps d'exécution diminuent. De plus, lorsque le nombre de threads est assez grand, on voit que les écarts entre les temps d'exécutions se réduisent. On remarque également que les résultats obtenus pour le programme séquentiel et le programme avec un seul thread (k=1) sont très similaires.

5. Tâche 5

a) Accélérer la boucle interne

Pour $i = 2$, il faut mettre à jour tous les nombres pairs (2 exclu) qui ne sont pas premiers, donc nous avons utilisé un pas de 2.

Pour $i \neq 2$ et i un nombre premier impair, nous avons utilisé un pas pair de 2i afin d'ignorer tous les nombres pairs (qui sont déjà évalués) lors du premier parcours de la boucle interne.

b) Réduction de l'espace mémoire

En éliminant tous les nombres pairs, on réduit de moitié l'espace du tableau de int.

Pour le cas de $4\,000\,000$, nous économisons ainsi $2\,000\,000 - 1$ *int*. Sachant qu'un *int* coûte $32 - bit$ d'espace, nous économisons ainsi $(2\,000\,000 - 1) * 32 = 63\,999\,968$ bits soit quasiment 8Mo de mémoire.

Remarque: nous pouvons encore améliorer l'espace mémoire en préférant un type tel *short int* qui diviserait l'espace mémoire du tableau par 2.

Tableau comparaison $N = 4000000$, *nombre de threads* $= 1, 2, 3, 4, 5, 67$ entre le programme sans optimisation (*tache3.c*), optimisation de la boucle interne (*tache5a.c*) et optimisation de l'espace mémoire (*tache5b.c*).

Nombre de threads	Tache3.c (non optimisé)	Tache5a.c (optimisation de la boucle interne)	Tache5b.c (optimisation de l'espace mémoire)
1	0,11702414	0.09277291	0.06758135
2	0,08371081	0.07247445	0.05261473
3	0,07129905	0.06379024	0.03846325
4	0,06499469	0.05760941	0.03442958
5	0,06186791	0.05512314	0.03543877
6	0,05968006	0.05082227	0.03749654
7	0,05702351	0.05108763	0.03452578

D'après le tableau au-dessus, le programme *tache5b.c* est la solution plus optimale.