Programmation parallèle	**CHPS0711**
Travaux Dirigés 2	**2018/2019**
Programmation par passage de messages – Le crible d'Érathosthène	**Pierre Delisle**

L'algorithme du crible d'Ératosthène, inventé par le mathématicien grec Ératosthène (276-194 av. J.-C.), permet de trouver les nombres premiers compris entre 2 et n. En voici le pseudo-code :

1. Créer une liste d'entiers (2, 3, 4, ..., n) sans les marquer
2. Initialiser k à 2, le premier entier non marqué de la liste
3. Répéter
   (a) Marquer tous les multiples de k entre k^2 et n
   (b) Trouver le plus petit entier plus grand que k qui est non marqué et initialiser k à cette nouvelle valeur
   Jusqu'à ce que $k^2 > n$
4. Les entiers non marqués sont des nombres premiers

Pour implémenter cet algorithme en C, nous allons utiliser un tableau de n-1 entiers (avec indices 0, 1, ..., n-2) pour représenter les entiers 2, 3, ..., n. La valeur booléenne (0 ou 1) à l'indice i indique que le nombre i + 2 est marqué. Le code C de notre implémentation est donné à la page suivante. Pour simplifier les choses, nous allons supposer que le nombre de nombres premiers initial (n - 1) est toujours un multiple du nombre de processeurs (p).

L'objectif de cet exercice est de proposer une parallélisation de ce code. Il faut tout d'abord déterminer de quelle façon nous allons décomposer le tableau de nombres pour le répartir aux processeurs.

Question 1 : Considérons une décomposition cyclique (approche modulo) des tâches aux processeurs. Dans ce cas, le processeur 0 est responsable des nombres (2, 2 + p, 2 + 2p, ...), le processeur 1 est responsable des nombres (3, 3 + p, 3 + 2p, ...), et ainsi de suite. Pouvez-vous identifier pourquoi cette approche n'est pas convenable ?

Nous allons plutôt utiliser une approche de décomposition en blocs. Il s'agit de diviser le tableau en p blocs contigus de taille égale de sorte à ce que le processeur 0 traite les n/p premiers entiers, le processeur 2 traite n/p suivants, et ainsi de suite.

Question 2 : Si n est le plus grand nombre premier à vérifier et p le nombre de processeurs,
- **(a) comment déterminer le plus petit nombre traité par le processeur i ?**
- **(b) comment déterminer le plus grand nombre traité par le processeur i ?**
- **(c) comment déterminer la taille du tableau de chaque processeur ?**

Question 3 : Comme nous arrêtons le marquage des nombre premiers lorsque $k^2 > n$, le plus grand entier utilisé pour le marquage est $\sqrt{n}$. Dans quelles circonstances pourrons nous assumer que tous les entiers utilisés pour le marquage seront stockés sur le processeur 0 ? Cette assomption vous semble-t-elle raisonnable ? Quelle en est la conséquence sur la complexité de l'algorithme ?

À l'intérieur de la boucle do...while du code donné à la page précédente, on trouve l'indice du 1er entier à marquer (k^2) et on marque ensuite tous les multiples de k jusqu'à la fin du tableau. Lors de la première itération de l'algorithme, tous les processeurs peuvent déterminer que k vaut 2.

Question 4 : En parallèle, comment déterminer l'indice du 1er entier à marquer pour chaque processeur lorsque k vaut 2 ? Et pour n'importe quel k ?

Après marquage des multiples de k, on détermine l'indice du prochain entier non marqué dans le tableau et on en déduit la prochaine valeur de k qui sera utilisée à l'itération suivante.

Question 5 : En parallèle, tous les processeurs seront-ils en mesure de déterminer le prochain k ? Comment faudra-t-il procéder ?

Une fois la boucle do...while terminée, on compte le nombre d'entiers non marqués, donc le nombre de nombres premiers, et on affiche ce nombre.

Question 6 : En parallèle, comment pourra-t-on effectuer ce compte ? Quelle(s) opération(s) seront nécessaire(s) ?

Question 7 : En fonction des réponses données dans les questions précédentes, écrivez le code C/MPI pour effectuer le crible d'Ératosthène en parallèle.

****

```
/*  Crible d'Eratosthene - Pierre Delisle */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {
    int n;              /* Dernier nombre premier a verifier (2..n)*/
    int taille;         /* Taille du tableau de nombres premiers a verifier */
    int *marques;       /* Tableau permettant de marquer les nombres verifies */
    int indexMarques;   /* Index du nb premier courant dans le tableau "marques" */
    int premierCourant;/* Nombre premier courant */
    int compte;             /* Compteur de nombres premiers */
    int indDebut;       /* Indicateur de debut de marquage*/
    int i;

    if (argc != 2) {  /* Verifie que un seul argument a ete fourni */
        printf ("Usage : %s n\n", argv[0]);
        exit(1);
    }
    n = atoi(argv[1]);

    taille = n - 1;
    marques = (int *) malloc (sizeof(int) * taille);
    for (i = 0; i < taille; i++)
        marques[i] = 0;
    indexMarques = 0;
    premierCourant = 2;

    do {
        indDebut = premierCourant * premierCourant - 2; /* indice du 1er entier a marquer (k2)*/
        for (i = indDebut; i < taille; i+= premierCourant) /* Pour tous les multiples du nb premier, */
            marques[i] = 1;                                 /* on marque le nombre*/
        while (marques[++indexMarques]);  /* On trouve l'indice de la prochaine case non marquee */
        premierCourant = indexMarques + 2;  /* ce qui donne le nb premier du prochain tour */
    } while (premierCourant * premierCourant <= n);  /* On arrete si son carre est > que n*/

    compte = 0;
    for (i = 0; i < taille; i++)
        if (!marques[i])
            compte++;
    printf("Il y a %d nombres premiers plus petits ou egaux a %d\n", compte, n);
    printf("Ces nombres sont : ");
    for (i = 0; i < taille; i++)
        if (!marques[i])
            printf("%d ", i+2);
    printf("\n");

    free (marques);
  return 0;
}
```