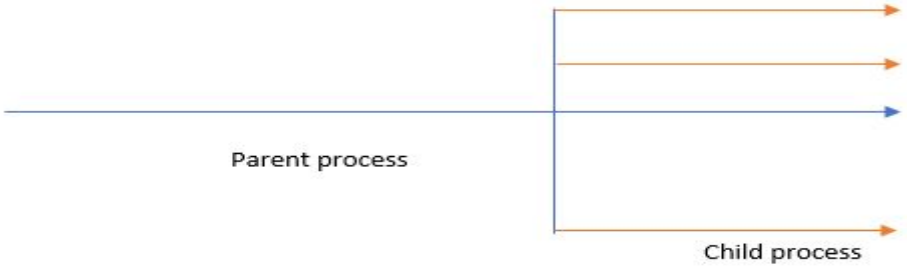
Nacir Bouazizi
Triet Tran
4A STI BIG Data

Rapport du projet HPC

1. what method you choose to go parallel ?

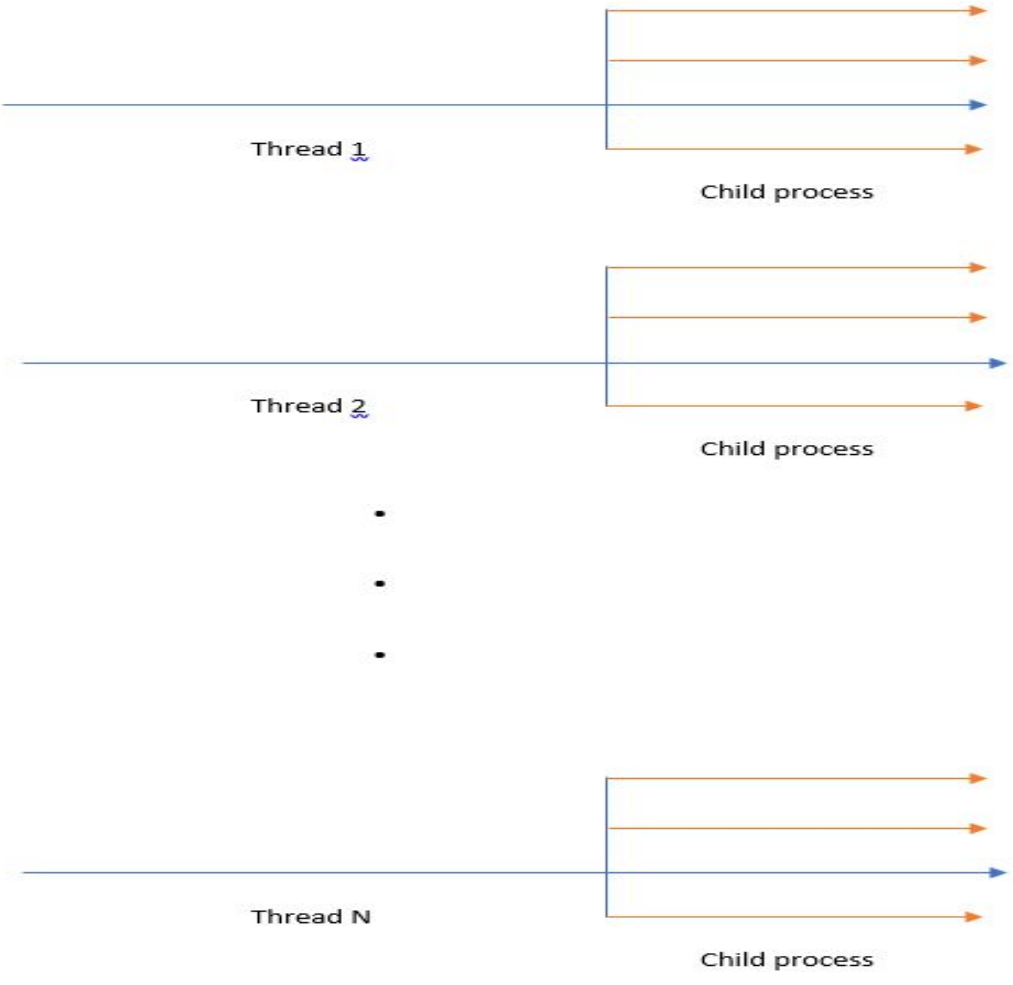
Mono Processeur :

L'idée de monoprocesseur: on utilise 1 coeur. Le processus parent va créer des processus fils pour paralléliser le tâche.



Multi Processeur :

On va utiliser N threads qui vont exécuter en même temps et chaque thread va exécuter la commande system M fois.



Etapes :

On a choisi de **paralléliser** la partie de lecture du fichier qui contient les hash + mot de passe en claire (avec grep).

-> Cette partie du code : (on va la paralléliser)

```
  for ( ; current_checker_running < MAX_FILS ; )
  { // forking until reaching MAX_FILS sons
        if ((n=fork())<0)
                {
                        perror("fork error");
                        exit(1);
                }
        if (n!=0) {//Father code, so I'm couting one more son

                current_checker_running++;
                printf("\n[INFO] Started  %dth son searching for password %s\n", current_checker_running, current_password_to_analyse);
password_to_analyse = readline(ds);

                  }
                else { // son code.
                        execl("/bin/grep", "grep", current_password_to_analyse, dict_file ,NULL);
                       // There's no return from execl, remember it ?
                  }
```

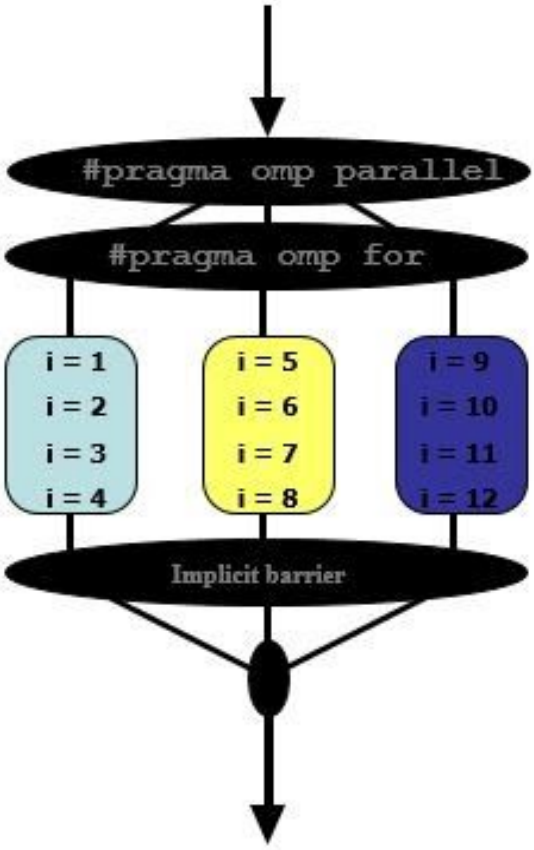
2. Pour le parallélisme : on a utilisé openMP.

Avant de passer à l'explication de notre code on commence par expliquer comment on peut paralléliser la "boucle for" en utilisant openMP.

Exemple simple pour calculer somme de tableaux element par element :

```
#pragma omp parallel

#pragma omp for

for(i = 1; i < 13; i++)

{

        c[i] = a[i] + b[i];

}
```

Supposons qu'on a 3 threads qui vont s'exécuter en parallèle le schéma suivant illustre ce qui va se passer :



étape 1 : Les threads se voient attribuer un ensemble indépendant d'itérations

étape 2 : Les threads s'attendent à la fin d'exécution dans la barrière implicite.

on a utilisé cette approche pour paralléliser l'exécution de grep à chaque fois qu'un thread lit une ligne du fichier qui contient les hash.

notre code :

```
//set number of thread
nb_thread = atoi(argv[1]);
omp_set_num_threads(nb_thread);

#pragma omp parallel private(thread_id)
{

        #pragma omp for
        for (i=0; i<30; ++i)
          {
            line = readline(ds);
            char cmd[100]="";
            thread_id = omp_get_thread_num();
            //printf("thread=%d\tline=%s\n",thread_id,line);
            sprintf(cmd,"%s %s %s","grep",line,"dict_hashed.txt");
                system(cmd);
          }
}
```

- On prend le nombre de threads à exécuter en argument.

- La variable privé thread_id : le numéro associé à chaque thread.

- On fait 30 itération car le nombres de lignes dans le fichier "shadowSmall.txt" est 30. (le seul petit souci de notre code c'est qu'il faut savoir le nombre de lignes dans le fichier qui contient les hashs pour réussir à utiliser la loop for parallélisé de openMP).
- On fixe le nombre d'itération à 3000 si on veut utiliser le fichier "shadow.txt"

-> on remarque que chaque thread va lire une ligne du fichier qui contient les hashs et va après exécuter la commande grep pour faire du matching.

(le schéma précédent illustre plus le fonctionnement de la "for loop" parallélisée)

3. Evaluation des performances

Compilation :

```
triet@XPSTriet:~/Desktop/HPC/Project resources-20200401$ gcc -fopenmp -o openmp openmp_parallel.c
```

On utilise time pour mesurer les performances.

3.1) Évaluer le temps d'exécution sur le fichier "shadowSmall.txt", avec un nombre de thread égal à 1 et 4 :

```
triet@XPSTriet:~/Desktop/HPC/Project resources-20200401$ time ./openmp 1 dict_hashed.txt shadow.txt
approbation a9404b95b785c08214eae14195f829545163d15
assurer 9f3ff9e2bdeb7c821ed2f69c86a491a558cc11
breton 3b17fff19ad3f30c6194159845a3fcf932bc
brisant b86a584b6b1677881d468e1a2cf888fdeb888541
cachette 9f3854669b1d820f579c0ca7b43ade6d36799c8
chinois d156fff85db6e72c716ef16a3d968ef50a9b2dc
collation aee2faa2f3f246e441fe258e84a724ee788f649b
concis 6da6203cefecd9a107cac549b2ae899a9aa69e
corpus 62892ff8a183f45ee191a696796927ae39b2a64
enseigne 5ce2fa1480c0e1c6a12384c1f6a6d4c3969e32f5
foire 56494d1e213e192d57b60f613d8b1c1aaeb41c

real    0m0.176s
user    0m0.145s
sys     0m0.032s

triet@XPSTriet:~/Desktop/HPC/Project resources-20200401$ time ./openmp 4 dict_hashed.txt shadow.txt
approbation a9404b95b785c08214eae14195f829545163d15
assurer 9f3ff9e2bdeb7c821ed2f69c86a491a558cc11
breton 3b17fff19ad3f30c6194159845a3fcf932bc
brisant b86a584b6b1677881d468e1a2cf888fdeb888541
cachette 9f3854669b1d820f579c0ca7b43ade6d36799c8
chinois d156fff85db6e72c716ef16a3d968ef50a9b2dc
collation aee2faa2f3f246e441fe258e84a724ee788f649b
concis 6da6203cefecd9a107cac549b2ae899a9aa69e
corpus 62892ff8a183f45ee191a696796927ae39b2a64
enseigne 5ce2fa1480c0e1c6a12384c1f6a6d4c3969e32f5
foire 56494d1e213e192d57b60f613d8b1c1aaeb41c

real    0m0.041s
user    0m0.096s
sys     0m0.049s
```

résultat: Avec 1 thread, le temps d'exécution est d'environ 0.176s tandis qu'avec 4 thread, le temps d'exécution est d'environ 0.041s (4.3 fois plus vite qu'avec 1 thread)

3.2) Évaluer le temps d'exécution sur le fichier "shadow.txt", avec un nombre de thread égal à 1 et 8 :

```
triet@XPSTriet:~/Desktop/HPC/Project resources-20200401$ time ./openmp 1 dict_hashed.txt shadow.txt
inattention bca1d88b9cf5968c34c3796fbbc2c73e498e8
inspectorat 18602beeb09de6a135b5b8557cf922a19ad7627
langouste 6b2137a337cbba39fda2178a1fdd82bae51522c0
oubli bba1c7209f4fa81a1083365918c5d5c7afdcbf1
pester 3667b80422a2b2cce244dff49a751421c5b9f
proche b1214db6249bfc256b290a3f26ca3dded4b632
punissable b48a352a71d23101aa4428a59351b879589b846
radicales 2b7cb23926f9b565fbe3118f2073b19653a9cc1
roi 70fe19c13a407bd60b3b3ca92b8c191fba12e
sauvetage 709074c86b8494453c9d41f6d4d52deb6bd48a7d
serrer b594af7e4224bb613a19cdda6cf0801e281a128a
Sofres 8aa4ac4d524e76a86f41385e66427cbc593b
tels que 295017911b27ec37a8294c28828a89d9ea9e28c
tintement 05df827ff9afddd3bbfac842e3f261e98b682
vigueur a2baccd28f15b7fb1d17a1548a75dacf7e7e464
Vosges 53546a2e49145e3aeb4472db4fc2f32db34c2ec
agresseur 469e35298fbcd4702315e260bd5e68be7a5744
analogie 3e4eb73ffbd5045ed45325dd7a03c5f3d1ebeb2
Autrichiens fb7d196a4d671e0c2fb7e4863fe9774f394248e

real    0m9.206s
user    0m7.813s
sys     0m1.491s
```

```
triet@XPSTriet:~/Desktop/HPC/Project resources-20200401$ time ./openmp 8 dict_hashed.txt shadow.txt
punissable b48a352a71d23101aa4428a59351b879589b846
radicales 2b7cb23926f9b565fbe3118f2073b19653a9cc1
roi 70fe19c13a407bd60b3b3ca92b8c191fba12e
sauvetage 709074c86b8494453c9d41f6d4d52deb6bd48a7d
serrer b594af7e4224bb613a19cdda6cf0801e281a128a
Sofres 8aa4ac4d524e76a86f41385e66427cbc593b
tels que 295017911b27ec37a8294c28828a89d9ea9e28c
tintement 05df827ff9afddd3bbfac842e3f261e98b682
vigueur a2baccd28f15b7fb1d17a1548a75dacf7e7e464
Vosges 53546a2e49145e3aeb4472db4fc2f32db34c2ec
agresseur 469e35298fbcd4702315e260bd5e68be7a5744
analogie 3e4eb73ffbd5045ed45325dd7a03c5f3d1ebeb2
Autrichiens fb7d196a4d671e0c2fb7e4863fe9774f394248e

real    0m1.536s
user    0m8.281s
sys     0m1.351s
```

-> **Résultat**: Avec 1 thread, le temps d'exécution est d'environ 9.206s tandis qu'avec 8 thread le temps d'exécution est d'environ 1.536s (6.2 fois plus vite qu'avec 1 thread).

4. Annexes :

Notre code :

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <crypt.h>
#include <omp.h>
#include<time.h>

char * readline(FILE * f) {
   char * line = NULL;
        size_t len = 0;
   ssize_t read;
        if ((read = getline(&line, &len, f)) != -1) {
   line[read-1] = '\0';

        return line;
        }
   return NULL;
}

int main(int argc, char **argv)
{
   if(argc !=4) fprintf(stderr, "Usage:./go2 nb_thread dict_hashed
shadow\n", argv[0]), exit(EXIT_FAILURE);
   char * dict_file = argv[2];
   char * shasum_file = argv[3];
   FILE * ds = fopen(shasum_file, "r");
```

```
  int i, thread_id, nb_thread;
  char * line;

  //set number of thread
  nb_thread = atoi(argv[1]);
  omp_set_num_threads(nb_thread);

  #pragma omp parallel private(thread_id)
   {

      #pragma omp for
      for (i=0; i<3000; ++i)
       {
            line = readline(ds);
            char cmd[100]="";
            thread_id = omp_get_thread_num();
            //printf("thread=%d\tline=%s\n",thread_id,line);
            sprintf(cmd,"%s %s %s","grep",line,"dict_hashed.txt");
             system(cmd);
       }
   }
      fclose(ds);
 return 0;
}
```