Xavier Lambein
54621300

Projet 2 : Architecture

Victor Lecomte
65531300

LSINF1252

`TL;DR.` Nous avons choisi l'algorithme de divisions successives, sur lequel nous lançons plusieurs threads qui testent la division avec des valeurs de départ décalées.

Déroulement

Nous commençons par lire tous les entiers à factoriser et nous les plaçons dans un heap (dans l'ordre inversé, avec le plus petit au-dessus). Ensuite, tant que le heap n'est pas vide, nous enlevons le plus petit entier du heap et nous lançons dessus les n threads à notre disposition. Dès qu'ils trouvent un facteur, ils l'ajoutent au heap et divisent la valeur à factoriser.

Une fois que les n threads ont tous atteint la racine, cela signifie que le nombre restant est premier. Nous essayons de diviser chaque nombre dans le heap par ce facteur premier. Si aucun n'est divisible, ce nombre est la solution. Sinon nous continuons.

Voici le pseudocode pour le thread principal :

```
1  read all input integers
2  push them to heap
3
4  while heap is not empty:
5
6      get toFact from heap
7
8      for i in 1..n:
9          init thread i with (start=1+i, step=n)
10     end
11
12     for i in 1..n:
13         join thread i
14     end
15
16     divide heap by toFact
17
18     if no divison possible:
19         toFact is the solution
20         print it and exit
21     end
22
23 end
24
25 (if the loop ends input is invalid)
```

Remarquons que si la boucle `while` se finit, l'input est incorrect, car cela signifie qu'aucun facteur premier ne divise exactement un nombre exactement une fois.

Les n threads recherchent des facteurs du nombre courant `toFact` en parcourant tous les entiers de 2 à $\sqrt{\texttt{toFact}}$, avec un pas de n : le premier thread itérant sur $i = 2, 2+n, 2+2n, \ldots$; le second thread itérant sur $i = 3, 3+n, 3+2n, \ldots$; etc. Pour chaque i, le programme regarde s'il divise

`toFact`. Dans ce cas, il s'agit d'un facteur (pas nécessairement premier), et la procédure décrite ci-dessus est appliquée.

Voici le pseudocode pour les threads de recherche de facteurs :

```
1  for i = start, i*i <= toFact, i += step:
2
3      if i divides toFact:
4
5          lock mut_toFact
6              if i divides toFact:
7                  push i to heap
8                  divide toFact by i
9              end
10         unlock mut_toFact
11
12     end
13
14 end
```

Synchronisation des threads

Notre programme ne présente qu'une seule section critique à proprement parler : il s'agit du moment où un thread a trouvé un facteur de `toFact`. Celui-ci bloque alors une mutex qui empêche l'écriture sur la variable partagée `toFact`, ce qui permet au thread de diviser celle-ci par le facteur en toute sécurité. À l'intérieur de la section protégée par la mutex, le programme vérifie d'abord à nouveau si le facteur trouvé divise bel et bien `toFact`, pour être certain que la valeur de celui-ci n'aurait pas changée avant que la mutex ne soit bloquée.

Une deuxième zone de synchronisation apparaît lorsque tous les threads ont terminé de parcourir les entiers entre 2 et $\sqrt{\texttt{toFact}}$. Le thread principal est alors chargé de vérifier si une solution a été trouvée, et sinon de continuer l'algorithme avec le nombre suivant présent sur le heap. Il fait cela de manière très simple, en tuant les n threads, effectuant les vérifcations, puis éventuellement en relançant n nouveaux threads.