TP - LIF12 système d'exploitation

Première utilisation de threads

2015/09/15

I Début des threads

Vous devez paralléliser un petit programme : `lancement.cpp` sur le site de l'UE. Ce dernier répète un certains nombre de fois un calcul qui n'a pas d'importance. Le but de l'exercice est de savoir si vous pouvez paralléliser les appels à la fonction fct, récupérer le résultat (qui est le temps écoulé pendant le calcul d'après la fonction) et comparer ce temps avec le temps réellement écoulé.

Pour cela :

Q.I.1) - Dans la fonction main remplacez la boucle qui appelle nbthreads fois la fonction fct par le lancement de nbthreads threads qui exécutent la fonction.

Q.I.2) - Si cela ne vous est pas déjà arrivé, faite une erreur d'argument entre le lancement du thread et la fonction afin de vous habituer à lire les messages d'erreur de compilation.

Q.I.3) - Normalement, dès qu'il y a un nombre de threads important, vous pouvez observer que les messages affichés par ces derniers se mélangent. Sachant que les fonctions systèmes sont prévues pour fonctionner en environnement multithreads, pourquoi y a-t-il ce mélange et comment l'éviter ?

On observe le problème sur les appels chainés
cout << num << ": lancement de la fonction pour "<< nbtours << "itérations" et pas sur les printf. En fait, chaque appel gère correctement les affichages, mais un cout fait plusieurs affichages séparés. En fait il y en a 1 par <<. Les appels à l'opérateur << se mélangent d'autant plus que ce sont des appels systèmes donc qu'ils sont lent.
Pour éviter le problème, il faut construire la chaine de caractère a part et tout afficher d'un seul coup. C'est ce que fait fprintf. L'autre solution est d'utiliser les ostringstream pour construire la chaine avant de l'envoyer sur cout.

Q.I.4) - À la fin des threads obtenez le résultat de la fonction et affichez le temps que chaque thread pense avoir mis pour s'exécuter ainsi que la somme de ces valeurs.

La commande time permet de voir les différents temps utilisés par une commande :

```
$ time ./lancement.exxb 20
Th principale : lancement de 10 fois la fonction
0 lancement de la fonction pour 5 itérations
...
Th principale : Le temps total de calcul est 9.87181

real    0m9.876s
user    0m9.863s
sys     0m0.002s
```

- *real* est le temps écoulé pendant le calcul
- *user* est la somme des temps processeur utilisés par les threads en mode utilisateur (pour faire les calculs eux même)
- *sys* est la somme des temps processeur passés en mode noyau (pour faire les opérations systèmes comme les affichages).

Q.I.5) - Faites plusieurs essais avec 2, 3, 4, 8, 10, 20 threads.

5(a) - Pourquoi le temps utilisateur est-il supérieur au temps réel ?

5(b) - Le somme des temps que vous calculez est-elle liée au temps réel ? Au temps utilisateur ? Au temps système ?

5(c) - Est-elle identique, s'il y a une différence pourquoi ?

Pour la dernière question il faut comparer cela avec le nombre de processeurs de l'ordinateur.

Si votre ordinateur est multiprocesseur, les threads permettent un véritable parallélisme et le temps réel est donc inférieur à la somme des temps processeur nécessaires.
Le temps mesuré en utilisant gettimeofday est le temps écoulé entre 2 moments. Il tient compte des autres threads ou processus qui fonctionnent sur la machine et donc peut être très supérieur au temps de calcul effectué (le temps user). C'est le cas si le thread doit attendre pour utiliser le processeur, donc s'il y a plus de threads qui calculent que de processeurs.
Selon le nombre de processeurs de votre machine, vous devriez donc observer que la somme des temps vu par les threads est a peut près celle du temps utilisateur lorsqu'il y a moins de threads que d'unité de calcul. En effet, ces derniers trouvent toujours un processeurs disponible pour faire les calculs. Ils n'attendent jamais.
Dès que le nombre de threads est supérieurs au nombre de processeurs, le temps mesuré devient plus important que le temps de calcul car il tient compte des moment où les threads étaient en attente du processeur.
Vous pouvez consulter les images obtenues sur un ordinateur avec 8 coeurs de calcul (attention, il n'y a qu'un test à chaque fois, les résultat sont assez imprécis).

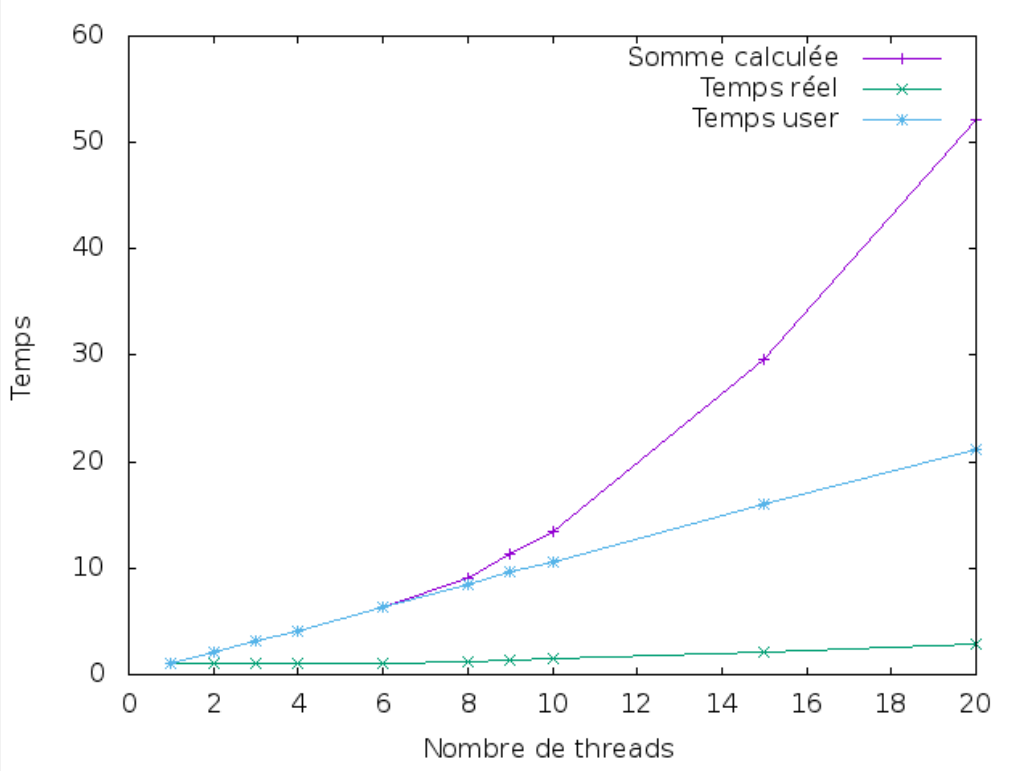

Figure 1 – Temps mesuré en fonction du nombre de threads

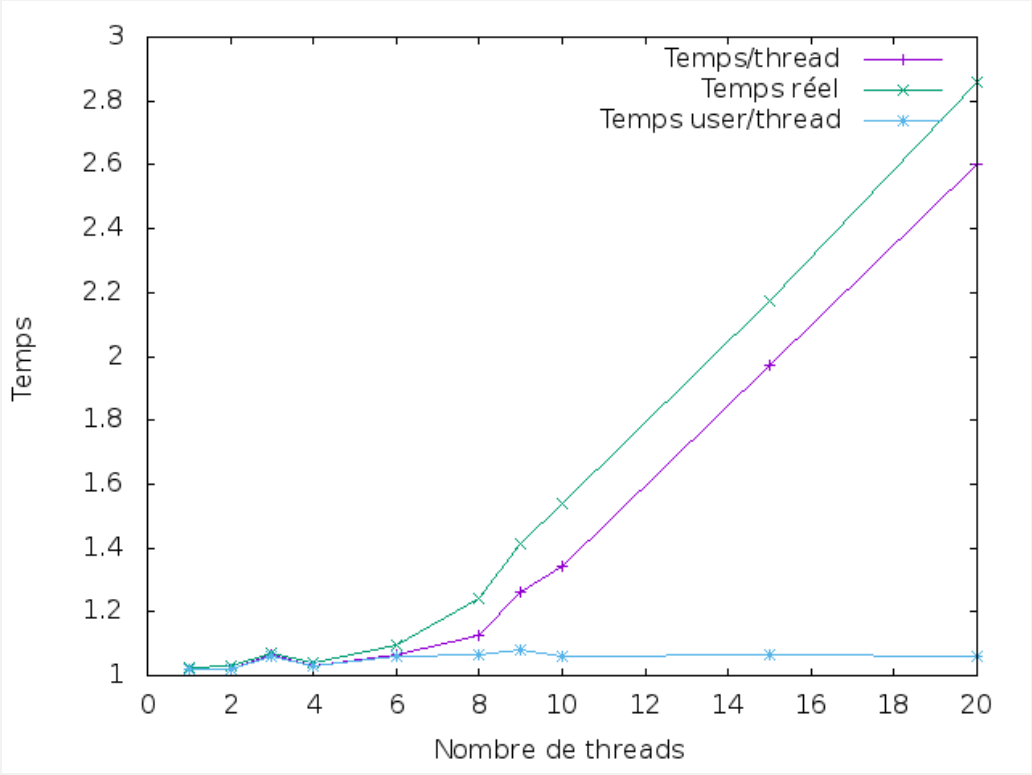

Figure 2 – Rapport du temps et du nombre de threads