OSEM

Compte Rendu TP1 – 12/10/16

David Toty (3000755) & Maxime Tran (3000738)

Dans ce TP nous allons nous intéresser aux différences de performances en matière de temps entre le nombre de threads alloués à un programme orienté multi-threadé.

Pour cela nous avons besoin d'un environnement mono-processeur car les machines à notre disposition, qui sont multiprocesseur, ne permettent pas une comparaison juste.

Nous allons utiliser un prototype virtuel de TSAR, nous allons pour cela télécharger un package dédié et fourni dans le cadre du TP.

Dans ce package, un fichier *SourceMe* permet d'initialiser quelques variables d'environnements :

$ source SourceMe

Maintenant nous pouvons lancer le prototype virtuel TSAR, dans le package *almos-tsar-mipsel-1.0/test/pf1* :

$ make sim1

Le but va être d'insérer notre propre application multi-threadé dans le prototype virtuel, pour cela nous allons avoir besoin de cross-compilé notre application précédemment écrite.

NOTRE APPLICATION :

Un squelette de l'application est fournie de base, ce squelette contient un main qui créer un nombre de threads déterminer et qui les attend.
Notre unique but va donc être de créer une application multi-threadé et de faire travailler les threads dessus.

Pour notre implémentation, le processus main va :

- Créer une chaîne globale de caractère aléatoire.
- Cette chaîne sera découper en fonction du nombre de threads
    - Chaque thread va analyser cette chaîne et compter une occurrence d'un caractère donner en argument.
        -

Pour plus de confort nous avons donner des arguments à notre programme :

ARG 1 = Longueur de la chaîne aléatoire créée
ARG 2 = Le char recherché
ARG 3 = Nombre de threads

Ainsi, une exécution de notre programme, qui se nomme hello, sera :
(Dans le prototype virtuel)

$ exec /bin/hello 100 a 4

Cela à pour effet de créer une chaîne aléatoire de 100 caractère, la chaîne sera découpé en 4 donc 25 char pour chaque threads, les threads vont chacun compter les occurrences de 'a' et vont renvoyer le résultat au processus main qui va additionner tous les résultats.

Pour cross-compiler, nous utilisons un makefile fourni dans *almos-tsar-mipsel-1.0/apps/hello_world*. Les commandes sont :

$ make TARGET=tsar
$ make install

Pour tester l'application sur l'environnement Linux, nous pouvons éventuellement utiliser la commande :

$ make TARGET=linux

Pour le temps nous alors utiliser des structures inclues dans time.h et l'utilisation de méthodes comme clock(), le temps sera mesuré entre la création de threads et la terminaisons de celles ci.

Il ne nous reste finalement qu'à exécuter le programme avec différent nombre de threads et de comparer les résultats.

ETUDES DES PERFORMANCES :

Le timer est placé avant le pthread_create et après le pthread_join :

Sim1 :

NbThreads	2	3	4	5
Temps (s)	**8,32**	**7,43**	**5,96**	**28,42**

Sim4 :

NbThreads	2	3	4	5

Temps (s)	**37,38**	**19,60**	**21,73**	

On voit que le speedup n'est pas forcement présent. Le placement du timer est donc à revoir, on le positionne maintenant entre le début et la fin de la fonction du thread :

Le timer est placé au début de la fonction thread_func, les threads envoient leurs leurs temps au processus principal qui fait la somme de tout les temps.
Sim1 :

NbThreads	2	3	4	5
Temps (s)	**0,058250**	**0,046215**	**0,031377**	**0,03919**