TD de Systèmes d'exploitation
François Cayre / `francois.cayre@grenoble-inp.fr`
Timothée Gerber / `timothee.gerber@gipsa-lab.fr`

Calcul matriciel parallèle

Notions abordées : tâches, exclusion mutuelle et synchronisation.

Fonctions POSIX associées : `pthread_create`, `pthread_join`, `pthread_mutex_lock`, `pthread_mutex_unlock`.

1 Calcul parallèle

Le but de ce TD est de procéder à la multiplication en parallèle de deux matrices `m1` et `m2`. La calcul parallèle vise à tirer parti des capacités des machines dotées de plusieurs cœurs d'exécution. Paralléliser un programme est un tâche ardue qui nécessite une compréhension profonde de l'algorithme à paralléliser, spécialement de son déroulement temporel en machine.

Le calcul parallèle introduit également la notion de **granularité** : la charge de travail que l'on confie à chaque cœur. Nous allons choisir la granularité la plus fine possible pour ce problème : une tâche ne sera chargée que d'incrémenter le terme courant de la matrice résultat avec chacun des produits des coefficients des matrices `m1` et `m2`.

Le terme courant de la matrice résultat s'écrit : $r_{i,j} = \sum_{k=1}^{N} m1_{i,k} \times m2_{k,j}$

Chacune des N tâches effectuera donc le calcul suivant : $r_{i,j} = r_{i,j} + m1_{i,k} \times m2_{k,j}$

Le problème qui se pose ici est double :

1. assurer que chaque tâche a bien accès à toutes les matrices ;
2. assurer que le calcul sera juste.

Le premier problème est résolu en utilisant précisément des tâches : toutes les matrices (`m1`, `m2` et `r`) seront dans l'espace mémoire du processus qui contient toutes les tâches.

Le second problème est clairement un problème d'exclusion mutuelle entre N tâches. Nous comparerons trois méthodes d'exclusion mutuelle : l'algorithme du boulanger de Leslie Lamport, l'utilisation d'un mutex POSIX, et celle d'une instruction atomique (*compare and swap*). On vérifiera que le calcul est faux en l'absence de synchronisation.

2 Machines multi-processeurs ?

Il ne vous aura pas échappé que les machines sur lesquelles vous devrez tester votre code doivent avoir au minimum deux cœurs physiques d'exécution...

3 Travail à réaliser

Le code de synchronisation des tâches effectuant le calcul parallèle est à compléter dans le fichier `pmul.c` .

La première étape est de créer et gérer les tâches dans la fonction `mat_pmul()` . Ces tâches travaillant dans la section critique, il faudra dans un second temps implémenter les méthodes de synchronisation pour éviter de fausser le calcul matriciel.

Vous trouverez les lignes où vous devrez insérer votre code en tapant un petit :

```
$ grep -n TODO *.c
```

4 Une fois que vous aurez terminé de coder

- Vérifiez que le résultat de la multiplication matricielle est correct, comparé à l'absence de synchronisation ;
- Comparez les temps d'exécution, affichés en micro-secondes – et discutez-les ! En particulier, vous argumenterez à l'aide des notions de granularité et de changement de contexte ;
- Proposez et codez un calcul matriciel parallèle efficace en choisissant une granularité qui vous semble davantage adaptée au problème.

5 Pour aller plus loin...

Algorithme du boulanger ?

http://research.microsoft.com/users/lamport/pubs/pubs.html#bakery

Le papier original de Lamport, avec la preuve de correction :

http://nob.cs.ucdavis.edu/classes/ecs150-1999-02/sync-bakery.html

À quoi sert *vraiment* la variable `choix` dans l'algorithme ?

Bon travail...