Travail pratique #1 — INF8541 (gr. 20)

Automne 2001

Date de remise : Au plus tard jeudi, 18 octobre, 9h00. Tout travail remis *après* l'heure indiquée sera considéré en retard (pénalité de 10 % par jour).
Aucun travail ne sera accepté après lundi, 22 octobre, 13h00 (parce que j'afficherai alors ma solution sur le *web*, pour discussion en classe au cours suivant).

Ce travail doit être fait de façon individuelle.

Le problème

Pour ce travail, vous devez développer un programme *Threaded-C* permettant de traiter un tableau (de longueur `n`) de nombres point-flottants (`float`) dans le but de calculer la moyenne des nombres de ce tableau pour ensuite produire un nouveau tableau contenant la différence entre le tableau initial et la moyenne calculée. Par exemple :

```
Entree~:
  a = {10.0, 20.0, 10.0, 20.0, 15.0}
  n = 5

Sortie~:
  b = {-5.0,  5.0, -5.0,  5.0,  0.0}
```

Pour le calcul de la moyenne, vous pouvez utiliser la stratégie que vous désirez, en autant qu'il s'agisse d'une stratégie parallèle. Par contre, pour la production du tableau `b` à partir de `a` et de la moyenne ayant été calculée, vous devez produire deux versions différentes :

1. Une première version permettra de calculer le tableau `b` en utilisant une stratégie de *parallélisme récursif*. Plus précisément, une approche diviser-pour-régner dichotomique (deux sous-problèmes) devra être utilisée, chaque appel récursif générant un appel récursif (`TOKEN`) indépendant.

   Toutefois, contrairement à une solution récursive simple où la décomposition se termine lorsque le problème est vraiment trivial — par ex., tableau de longueur `1` — vous devrez plutôt introduire un argument supplémentaire — appelé la *valeur de coupure* — indiquant à partir de quelle taille de problème la décomposition récursive *doit se terminer*. Par exemple, si la valeur de coupure est de `10`, alors lorsque la taille du tableau à traiter devient plus petite ou égale à `10`, la décomposition parallèle récursive se termine et une approche non-récursive (et séquentielle) est utilisée.

   Autre point, le traitement *ne doit pas se faire en place*; en d'autres mots, le tableau initial `a` ne doit pas être modifié.

2. Une deuxième version permettra de faire le traitement désiré en utilisant une approche de *parallélisme itératif* avec allocation statique des processus (en fonction du nombre de processeurs). Par exemple, si la machine fonctionne avec quatre (4) processeurs, alors les éléments du tableau à traiter seront simplement répartis, *de façon équilibrée*, entre les quatre processeurs. Chacun de ces sous-tableaux (on parle aussi de *tranche*) sera alors traité, de façon indépendante, à l'aide d'un algorithme séquentiel (itératif).

Note : Vous pouvez définir deux (2) programmes distincts, un programme pour chacune des versions, ou encore définir un unique programme qui appele la version sélectionnée (récursive ou itérative) à l'aide d'un argument fourni à l'appel du programme.

Quelques expériences

Lorsque vos procédures fonctionneront correctement, vous devrez ensuite effectuer un certain nombre d'expériences de façon à évaluer les deux stratégies de traitement de même que l'influence de différents paramètres :

1. Pour la version récursive exécutée sur un tableau comportant un petit nombre d'éléments vs. un plus grand nombre d'éléments (par ex., 1000 vs. 40000 ou plus), pour quelle valeur de coupure (par ex., parmi 10, 100 ou 1000) le programme est-il le plus efficace? Est-ce que le nombre de processeurs utilisés pour exécuter le programme (e.g., 1, 2, 4) a une influence? Expliquez brièvement vos résultats et conclusions.
2. Pour différents nombres d'éléments (1000, 10000, 40000), déterminez la courbe d'accélération de la version non-récursive en fonction du nombre de processeurs (1, 2, 4 ou 8). Expliquez vos résultats.
3. Pour un petit nombre d'éléments vs. un plus grand nombre (par ex., 1000 vs. 40000 ou plus), laquelle des versions (récursive ou itérative) est la plus efficace (en utilisant, pour la version récursive, le point de coupure optimal identifié plus haut)? Est-ce que cela dépend aussi du nombre de processeurs? Expliquez.

Ce que vous devez remettre

- Une copie papier de votre code Threaded-C, de même qu'une copie envoyée par courriel (ficher `tar` si vous avez plusieurs fichiers).
- Des "*preuves*" du bon fonctionnement de votre code (voir plus bas).
- Les résultats de vos expériences (préférablement des tableaux, même si faits à la main), et les discussions associées.

Bonus

- Un bonus ($\approx$ 10 %) sera accordé si le calcul de la moyenne suivi de la génération du tableau `b` se font (surtout pour la version itérative) sans générer de copies inutiles des sous-tableaux (pas facile ;(.

Quelques remarques

Nombre de processeurs utilisés dans les expériences : Vous n'avez pas besoin d'expérimenter avec 16 processeurs. Il arrive parfois que l'un ou l'autre des processeurs ne fonctionne pas ou que la communication avec l'un des processeurs est "fragile"; en utilisant un maximum de 8 processeurs et en utilisant la commande `test_slaves`, vous devriez pouvoir produire les résultats requis sans trop de problèmes. (Voir le fichier `A-LIRE` dans votre répertoire sur Earthquake.)

Génération des tableaux de nombres à utiliser dans vos tests : Ces tableaux de nombres devraient être générés de façon aléatoire. Pour générer un nombre point-flottant aléatoire compris entre `0.0` et `MAX-1`, il suffit d'utiliser l'expression suivante, où `MAX` est la version entière du nombre maximum désiré :

```
1.0 * (rand() % MAX)
```

Preuve de bon fonctionnement : Il est préférable, lorsqu'on désire tester le bon fonctionnement d'un programme, d'avoir une option d'exécution permettant de produire une réponse simple (de style "`OK!`" ou "`Pas OK!`") pour déterminer si le résultat est correct ou non. Par exemple, pour ce problème, on peut définir une procédure auxiliaire `verifier_resultat` qui sera appelé avec les tableaux `a`, `b` ainsi qu'avec la `moyenne` calculée et qui vérifiera si tout semble correct (la différence entre les deux tableaux est *presque égale* à la moyenne).