Travail pratique #1 — INF7235 (gr. 10)

Hiver 2017

Date de remise: lundi 6 mars (après la semaine de relâche), 13h30. Tout travail remis *après* l'heure indiquée sera considéré **en retard** (pénalité de 10 % par jour, complet ou partiel). **Aucun** travail ne sera accepté après vendredi 10 mars, 9h00.

Ce travail peut être fait **seul ou avec une (1) autre personne**.

Programmation parallèle avec variables partagées

1 Objectif

Le but de ce travail est de vous familiariser avec les principaux patrons de programmation parallèle *avec variables partagées*. Ce travail peut prendre deux formes, décrites aux sections 2.1 et 2.2.

2 Ce que vous devez faire

2.1 Mettre en oeuvre et comparer diverses solutions parallèles pour un problème de votre choix

Ce premier type de travail est («relativement») *ouvert* quant au choix du sujet. En d'autres mots, *c'est à vous* de préciser le problème sur lequel vous travaillerez, en me consultant pour faire approuver le problème, les grandes lignes de sa spécification, etc.

Ce premier type de travail est aussi ouvert quant au choix du langage utilisé — Ruby/PRuby, Java, C avec *threads* Posix, C++ avec *Threading Building Blocks* — *mais pas OpenMP/C!* La contrainte est que le langage utilisé doit supporter une approche de programmation parallèle **avec variables partagées**. De plus, le programme résultant doit pouvoir être exécuté, de façon parallèle, sur la machine `japet.labunix.uqam.ca`.

Pour un travail de ce type, vous devez développer et comparer différentes versions[1] d'un programme fonctionnant sur une machine multi-processeurs. Plus précisément, vous devez minimalement réaliser et comparer les versions suivantes :

a. Une solution séquentielle, qui servira de référence pour le calcul des accélérations ainsi que pour la vérification des résultats.

b. Une solution parallèle **à granularité grossière** avec association **statique** entre tâches et *threads*.

c. Une solution parallèle **à granularité moyenne ou grossière** avec association **dynamique** entre tâches et *threads*... *ou une autre méthode* **appropriée** *à votre problème*.

L'objectif est évidemment **d'essayer d'obtenir** une version parallèle avec des performances intéressantes, tant au niveau du temps d'exécution que de l'accélération. De plus, il s'agit aussi de déterminer quelle approche semble la plus appropriée pour le problème traité — quelle approche permet d'obtenir les meilleures performances, les meilleures accélérations? — et de **justifier et expliquer** pourquoi il en est ainsi pour votre problème.

[1] Il peut s'agir de programmes distincts ou d'un seul programme qui reçoit en argument (via un paramètre de la ligne de commande ou une variable d'environnement) le nom/numéro de la version à exécuter — cf. labos.

Quelques suggestions de sujets possibles

- Génération de fractales, e.g., ensemble de Mandelbrot [WA99, Sect. 3.2][MSM05, Sect. 4.4].
- Opérations arithmétiques avec entiers (ou réels) à grande précision.
- Transformations géométriques d'images [WA99, Sect. 3.2], «*region labeling*» [And00, Exercice 9.8].
- Simulation d'automates cellulaires (par ex., «Jeu de la vie») [And00, Sect. 9.2.2].
- Résolution de systèmes d'équations linéaires [And00, Sect. 11.3].
- Partitionnement de données avec l'algorithme des k-moyennes (*k-means clustering*)

  `https://en.wikipedia.org/wiki/K-means_clustering`
- Satisfaisabilité d'un circuit [Qui03, Sect. 4.4]
- Simulation d'interactions entre particules, par ex., mouvement de planètes [And00, Sect. 11.2][Pac11, Sect. 6.1] (un peu plus compliqué).

Note : Si vous avez d'autres idées pour un sujet possible, contactez-moi (courriel) ou venez me voir (en prenant rendez-vous par courriel) et on en discutera.

2.2 Modifier ou étendre la bibliothèque `PRuby`

Pour un travail de ce type, il s'agit de **modifier** ou de **développer une extension** pour la bibliothèque `PRuby`, par exemple :

- Mise en oeuvre des méthodes `peach`/`peach_index` et `pmap`/`preduce` pour les `Hash` — actuellement, ces méthodes ne s'appliquent qu'à des `Array` ou `Range`.
- Développement de méthodes permettant d'émuler l'approche `Map/Reduce` [DG08].
- Développement de méthodes permettant d'émuler (une version simplifiée) de l'API pour *Apache Storm*, approche fondée sur une forme de parallélisme de flux :

  `http://storm.apache.org/releases/2.0.0-SNAPSHOT/Concepts.html`

Remarque : Je suis ouvert à d'autres types de projets, pas nécessairement liés à Ruby/PRuby. Le critère : être en lien avec la programmation parallèle sur une machine **à mémoire partagée**.

3 Ce que vous devez remettre

a. Document papier — remis en main propre ou dans la chute de travaux du secrétariat — expliquant ce que vous avez fait :

(i) Une brève description du problème que vous avez traité.

(ii) Une description des approches que vous avez utilisées et mises en oeuvre pour résoudre le problème. Plus précisément, vous devez expliquer brièvement de quelle façon (**comment?**) fonctionnent vos solutions, notamment, **quels patrons d'algorithmes parallèles et quels patrons de programmation parallèle** vous avez utilisés. Vous devez aussi expliquer **pourquoi** vous avez choisi ces approches.

(iii) Une description de votre stratégie de tests : de quelle façon avez-vous procédé pour vérifier que vos programmes parallèles produisaient les bons résultats?

(iv) Des résultats expérimentaux — **minimalement, temps d'exécution et graphes d'accélération** — et une analyse de ces résultats.[2]

Ces résultats expérimentaux devraient (au minimum) présenter l'effet sur les performances de **varier le nombre de threads** ainsi que **varier la taille du problème**.

De plus, pour cette partie pour le travail du premier type, il faut aussi comparer vos solutions entre elles, et aussi expliquer, le cas échéant, pourquoi l'une des solutions semble meilleure que les autres, ou sinon pourquoi il n'y a pas de différence.

Note : Dans vos mesures, vous pouvez ignorer le temps pour effectuer la lecture (fichier ou `stdin`) des données et l'écriture (fichier ou `stdout`) des résultats.

Note : Lorsque vous présentez des résultats expérimentaux (temps d'exécution, accélération), utilisez un nombre *raisonnable* et *uniforme* de chiffres significatifs.

Contre-exemple :

Nb. procs.	**Temps (sec)**
1	1,209
2	2,1
...	...
16	0,234597

Exemple :

Nb. procs.	**Temps (sec)**
1	1,21
2	2,10
...	...
16	0,23

b. Code source (*listing* papier) pour le code que vous aurez développé, y compris les tests.

Pour les travaux du premier type (2.1 Mettre en oeuvre et comparer...), votre «code source» *doit inclure un fichier* `makefile` définissant minimalement les deux cibles suivantes :

(i) `make tests` : Pour vérifier le bon fonctionnement de votre programme.

(ii) `make mesures` : Pour exécuter votre programme pour en mesurer les performances.

[2] Pour les temps d'exécution, vous pouvez présenter un tableau ou un graphe. Par contre, pour les accélérations, vous devez nécessairement présenter **un graphe d'accélération**.

La *qualité* (style) de votre code — présentation, clarté et simplicité, respect des principes DRY et KISS[3], structure, choix des identificateurs, respect du style propre au langage choisi, etc. — sera évaluée.[4]

Au sujet de la qualité du code, voir aussi les notes de cours, annexe «C. Style de programmation et qualité du code».

`http://www.labunix.uqam.ca/~tremblay/INF7235/Chapitres/chC-kiss-dry.pdf`

Remise électronique du code source

Vous devrez remettre **une version électronique** de votre code source (y compris les tests) à l'aide de l'outil Oto. Par exemple, supposons que `Devoir1` est le répertoire contenant le code source de votre projet et `ABCD11111101` est votre code permanent. Alors vous devrez exécuter la commande suivante sur `japet` **à partir du répertoire parent de** `Devoir1`:

```
$ ssh oto.labunix.uqam.ca oto rendre_tp tremblay_gu INF7235 ABC11111101 $(pwd)/Devoir1
```

Note : Pour les équipes de deux personnes, il suffit de séparer les deux codes permanents par une «,» (sans espace).

Pour vérifier que la remise a été effectuée correctement, il suffit ensuite d'exécuter la commande suivante (à partir du même compte usager) :

```
$ ssh oto.labunix.uqam.ca oto confirmer_remise tremblay_gu INF7235 ABC11111101
```

c. Preuve du bon fonctionnement de vos programmes : spécifications claires et explicites des tests, résultats obtenus corrects, résultats équivalents pour les diverses mises en oeuvre.

Par preuve de bon fonctionnement, j'entends le fait d'exécuter votre programme avec certaines données de tests (pas nécessairement les mêmes que celles pour mesurer les performances) et de pouvoir vérifier *de façon automatique* que les résultats produits sont bien ceux attendus. **Au minimum**, *vous devez vous assurer que les solutions parallèles produisent les mêmes résultats que la solution séquentielle.*

[3] DRY = «*Don't Repeat Yourself*» ; KISS = «*Keep It Simple, Stupid*».

[4] Une façon simple et directe d'exprimer l'aspect «qualité du code» est celle décrite par l'utilisation de la métrique *WTF/m* : `http://www.osnews.com/story/19266/WTFs_m`. Plus cette métrique est élevée, moins le code est de bonne qualité.

4 Critères de correction

Pour la remise de votre travail, vous devez utiliser **la feuille de présentation disponible à la page suivante**. Voir cette page pour le barème de correction.

Note : Le terme «respect» dans l'item «Bon choix d'approche et respect de cette approche» signifie que l'approche que vous avez mise en oeuvre **est bien celle que vous avez décrite, et justifiée**, dans la description de vos solutions.

5 Citation des sources

L'utilisation textuelle (directe ou traduite) d'une partie de texte sans une référence appropriée constitue *une forme de plagiat* — donc une **infraction de nature académique**. Pour plus d'informations sur ce qui est considéré comme du plagiat, consultez l'URL suivant — qui explique aussi ce qu'est une *reformulation* correcte :

`http://www.bibliotheques.uqam.ca/plagiat`.

Donc, notamment pour la partie où vous décrivez le problème et vos solutions, vous pouvez vous «inspirer» de diverses sources, mais si vous utilisez des *extraits* de ces sources, alors **il faut indiquer qu'il s'agit de citations et indiquer clairement et explicitement les sources**.

Références

[And00] G.R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming.* Addison-Wesley, Reading, MA, 2000.

[DG08] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[MSM05] T.G. Mattson, B.A. Sanders, and B.L. Massingill. *Patterns for Parallel Programming.* Addison-Wesley, 2005.

[Pac11] P.S. Pacheco. *An Introduction to Parallel Programming.* Morgan Kaufman Publ., 2011.

[Qui03] M.J. Quinn. *Parallel Programming in C with MPI and OpenMP.* McGraw-Hill, 2003.

[WA99] B. Wilkinson and M. Allen. *Parallel Programming—Techniques and Applications Using Networked Workstations and Parallel Computers.* Prentice-Hall, 1999.

Travail remis à Guy Tremblay

INF7235-10 : Devoir 1

À remettre lundi 6 mars (après la semaine de relâche), 13h30

Ne pas mettre dans une enveloppe

Nom	
Prénom	
Code permanent	
Courriel	

Nom	
Prénom	
Code permanent	
Courriel	

Description du problème, des solutions choisies et justification, de la stratégie de tests, conclusion		10 pts
Qualité générale du code		10 pts
Bon choix d'approches et respect de ces approches		10 pts
Résultats expérimentaux et analyse		10 pts
Bon fonctionnement et preuve de bon fonctionnement (dont qualité des tests)		10 pts
Qualité du français		5 pts
(Bonus) Difficulté, originalité du problème choisi		5 pts
Total		55 pts

Note globale :	/ 10