Travail pratique #2 — INF7235-10

Hiver 2016

Date de remise: vendredi, 29 avril, 9h00.
Aucun travail ne sera accepté après lundi, 2 mai, 10h00.

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

Programmation parallèle par échange de messages avec MPI

Objectif

Dans le devoir 1, vous avez approfondi l'utilisation de paradigmes de la programmation parallèle *avec variables partagées*. Dans le devoir 2, vous allez approfondir l'utilisation de paradigmes et techniques de programmation parallèle **par échange de messages** avec MPI [For94].

Comme pour le devoir précédent, ce devoir est («relativement») *ouvert* quant au choix du sujet. Donc, *c'est à vous* de préciser votre projet, en me consultant pour faire approuver le problème, les grandes lignes de sa spécification, etc.

Types de projets

Votre projet pourrait être de l'un des deux types suivants ($\star$) :

- **Utilisation de MPI sur une grappe de calcul** : Pour un projet de ce type, vous devez développer et comparer deux (2) versions d'un programme parallèle utilisant MPI et fonctionnant sur une grappe de calcul. Il peut s'agit d'une unique programme, qui exécute diverses versions en fonction des arguments reçus, ou de programmes distincts.

  L'objectif est évidemment d'obtenir un programme parallèle avec des performances intéressantes — temps d'exécution, accélération et efficacité. De plus, il s'agit aussi de déterminer quelle(s) approche(s) ou quelle(s) technique(s) de programmation MPI semble(nt) la(les) plus appropriée(s) pour le problème traité — en d'autres mots, quelle approche ou technique permet d'obtenir les meilleures performances.

  Les versions parallèles que vous développerez et comparerez peuvent différer au niveau du paradigme algorithmique utilisé — parallélisme de données, sac distribué de tâches, algorithme systolique, algorithme avec pipeline [And00, Chap. 9][MSM05, Section 4.6 et 5.4][Chap. 21 des notes de cours] — ou au niveau de la stratégie pour la décomposition du problème et/ou de l'agglomération des tâches. Vos deux solutions peuvent aussi différer au niveau plus «technique» — utilisation de différentes distributions des données, (par ex., décomposition par groupe de lignes vs. par blocs), utilisation de différents styles d'opérations de communication (par ex., opération point à point vs. opération collective, opérations synchrones vs. asynchrones), etc.

  Note : Dans tous les cas, vous devriez vous inspirer de l'approche PCAM de Foster [Fos95] pour décrire et justifier les grandes lignes de votre solution algorithmique.

  Quelques exemples possibles de problèmes :

  - Multiplications d'entiers à grande précision (ou de polynomes).
  - «*Region labeling*» d'images [And00, Exercice 9.8].
  - Solution itérative à l'équation de diffusion à deux dimensions [And00, Sect. 11.1].
  - Simulation d'automates cellulaires (jeu de la vie ou autre) [And00, Sect. 9.2.2].
  - Calculs d'interactions entre particules [And00, Sect. 11.2].
  - Tri d'une série de nombres [Qui03, Chap. 14].
  - Résolution d'un système 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`

- **Mise en oeuvre de MPI** : Pour un tel projet, il s'agit de développer une mise en oeuvre (évidemment simplifiée!) de MPI — un *binding* MPI — pour un langage de votre choix. Cette bibliothèque MPI devra pouvoir fonctionner soit sur un *cluster*, soit sur une machine multi-coeurs ou multi-processeurs (simulation sur une machine à mémoire partagée).

  Pour un tel projet, il s'agit donc de développer une mise en oeuvre d'une bibliothèque de style MPI, plutôt que d'utiliser MPI pour obtenir un programme parallèle.

($\star$) Note : Si vous avez d'autres idées pour un projet, venez me voir et on en discutera. Par contre, le projet doit avoir un lien avec la programmation parallèle *par échange de messages* avec **processus** (poids léger ou poids lourd) — donc sans mémoire partagée.

Ce que vous devez remettre

1. Un rapport écrit expliquant ce que vous avez fait :

   - **Utilisation de MPI sur une grappe de calcul** :
     (a) Une description du problème que vous avez traité.
     (b) Une description des deux (2) approches que vous avez utilisées et mises en oeuvre pour résoudre le problème, par exemple, une description à l'aide de pseudocode, à l'aide de l'approche PCAM, en termes des stratégies ou patrons d'algorithmes parallèles (cf. Chap. 5 des notes de cours). Bref, vous devez expliquer de quelle façon vos solutions fonctionnent... et comment elles diffèrent.
     (Formulé autrement : je veux avoir une bonne idée de comment vos programmes fonctionnent — et diffèrent — **avant** d'aller voir le code source!)
     (c) Une brève description de votre stratégie de tests — voir plus bas.
     (d) Des résultats expérimentaux — **minimalement : temps d'exécution et graphe d'accélérations** — et une analyse de ces résultats.
     Pour cette partie, il faut évidemment *comparer vos solutions entre elles* ainsi qu'avec une solution séquentielle — accélération absolue. Le cas échéant, vous devez aussi expliquer pourquoi l'une des solutions semble meilleure que l'autre. Finalement, pour les résultats expérimentaux, il est important de faire varier la taille du problème, car les résultats qui nous intéressent (accélérations) dépendent souvent de ce facteur.

     En d'autres mots, vos résultats expérimentaux devraient (au minimum) présenter l'effet sur les performances de varier **le nombre de processus** ainsi que **la taille du problème**.

     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 vos données (temps d'exécution, accélération ou efficacité), 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

     (e) Une conclusion où vous discutez brièvement de votre expérience de programmation en MPI : problèmes et difficultés rencontrés avec le langage, l'environnement, choses intéressantes apprises, etc.

   - **Mise en oeuvre de MPI** : Un rapport (manuel *technique*) expliquant les grandes lignes de la mise en oeuvre de votre bibliothèque style MPI : Quelles opérations ont été mises en oeuvre? Comment la bibliothèque fonctionne-t-elle? Quelles sont les limites et contraintes? Quelles difficultés avez-vous rencontrées? Etc.

2. Code source pour le/les programme/s que vous aurez développé/s, y compris les tests.

   Votre «code source» *doit inclure un fichier* `makefile` *approprié*, définissant minimalement les deux cibles suivantes :

   (a) `make tests` : Commande qui permet de vérifier le bon fonctionnement de vos solutions.

   (b) `make mesures` : Commande qui permet d'exécuter vos solutions pour en mesurer les performances.

   La *qualité* (style) de votre code — présentation, clarté et simplicité, respect des principes DRY et KISS[1], structure, choix des identificateurs, etc. — sera évaluée.

3. Preuve du bon fonctionnement de vos programmes MPI — spécifications claires des tests, résultats obtenus corrects, résultats équivalents pour les deux programmes, etc. — ou de votre bibliothèque style MPI — à l'aide de divers exemples de programmes.

   Dans le cas de programmes MPI (premier type de projet), par preuve de bon fonctionnement j'entends plus spécifiquement le fait d'exécuter votre programme avec certaines données de tests (pas nécessairement les mêmes que celles pour mesurer les performances) et de vérifier *de façon automatique* — idéalement, *avec des assertions* — que le résultat produit est bien celui attendu.

   Notamment, vous pouvez vérifier que les résultats produits par les versions parallèles sont identiques entre eux, et identiques à ceux produits par la version séquentielle — **ceci est, assurément, le minimum que vous devriez faire!**

Remise électronique du rapport et du code source

Vous devrez remettre **votre rapport** **en format PDF** et votre code source (y compris les tests) sous forme électronique, à l'aide de l'outil Oto.

Par exemple, supposons que `Devoir2` est le répertoire contenant le rapport et le code source de votre projet et `ABCD11111101` est votre code permanent. Alors vous devrez exécuter la commande suivante — sur `japet` ou `malt`, mais pas `turing` — **à partir du répertoire parent de** `Devoir2`:

```
$ ssh oto.labunix.uqam.ca oto rendre_tp tremblay_gu INF7235 ABC11111101 $(PWD)/Devoir2
```

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
```

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

Critères de correction

Pour la correction, j'utiliserai le barême dans **la grille** disponible à la page 6.

Note : Cette grille s'applique plus spécifiquement au premier type de projet. Pour le deuxième type, l'aspect «choix d'approches» sera interprété comme les stratégies choisies pour la mise en oeuvre.

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 très bien ce qu'est une *reformulation* correcte :

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

Donc, notamment pour la partie où vous décrivez le problème et les grandes lignes de votre solution, vous pouvez vous «inspirer» de diverses sources, mais si vous utilisez des *extraits* de ces sources, **alors il faut indiquer qu'il s'agit d'une citation** et indiquer explicitement la source.

Références

[And00] G.R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming.* Addison-Wesley, Reading, MA, 2000. [QA76.58A57 2000].

[For94] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4), 1994.

[Fos95] I. Foster. *Designing and Building Parallel Programs.* Addison-Wesley, 1995. `http://www-unix.mcs.anl.gov/dbpp`.

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

[Pac97] P.S. Pacheco. *Parallel Programming with MPI.* Morgan Kaufman, 1997.

[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.

Note : J'ai des copies de tous ces livres et je peux les prêter, pour quelques jours. Il faut toutefois me le demander à l'avance, par courriel, car la plupart sont à la maison.

Travail remis à Guy Tremblay

INF7235-10 : Devoir 2

À remettre au plus tard le vendredi, 29 avril, 9h00

Remarque : Contrairement au devoir 1, le rapport et le code source doivent être remis sous forme électronique

---

Nom	
Prénom	
Code permanent	
Courriel	
Nom	
Prénom	
Code permanent	
Courriel	

Description du problème, des solutions choisies et de leurs différences, description 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
Forme et qualité du français		5 pts
(Bonus) Difficulté, originalité du problème choisi, des solutions utilisées		5 pts
Total		55 pts

Note globale : / 10