Travail pratique #2 — INF7235-10

Hiver 2014

Date de remise: mercredi, 23 avril, 16h00.
Aucun travail ne sera accepté après lundi, 28 avril, 16h00.

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 vous êtes familiarisés avec les paradigmes de la programmation parallèle *avec variables partagées*. Dans le devoir 2, vous allez plutôt vous familiariser avec les paradigmes de programmation parallèle *par échange de messages*, plus précisément, selon le modèle 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 :

- **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.

  Pour un tel projet, l'objectif est évidemment d'obtenir un programme parallèle avec des performances intéressantes, par exemple, au niveau du temps d'exécution, de l'accélération et de l'efficacité. De plus, il s'agit aussi de déterminer quel paradigme — parallélisme de données, sac distribué de tâches, algorithme systolique, algorithme avec pipeline [And00, Chap. 9][MSM05, Section 4.6 et 5.4] ou autres — semble le plus approprié pour le problème traité (lequel permet d'obtenir les meilleures performances). Vous pouvez (devriez) aussi vous inspirer de l'approche PCAM de I. Foster [Fos95] pour proposer vos **deux** stratégies. **En d'autres mots, l'objectif est de comparer deux programmes conçus avec des approches différentes, au niveau du paradigme ou de la stratégie pour la décomposition du problème et/ou de l'agglomération des tâches de base.** Il ne s'agit donc pas de comparer deux programmes conçus de la même façon (par ex., décomposition par blocs de lignes) mais utilisant des mises en oeuvre MPI différentes (par ex., opération point à point vs. opération collective).

  Quelques exemples possibles de problèmes :

  - Multiplications d'entiers à grande précision (ou de polynomes).
  - 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].

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

  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.

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* (mémoire distribuée).

Ce que vous devez remettre

1. Document 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 — en d'autres mots, vous devez expliquer brièvement de quelle façon vos programmes fonctionnent.
     (c) Des résultats expérimentaux — **minimalement : temps d'exécution et graphe d'accélérations (relatives)** — et une analyse de ces résultats. Pour cette partie, il faut évidemment *comparer vos solutions entre elles* et, le cas échéant, expliquer pourquoi l'une des solutions semble meilleure que l'autre. Il est aussi 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 résultats expérimentaux (temps d'exécution, accélération, 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

     (d) Une description de votre stratégie de tests — voir plus bas.
   - **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 (*listing* papier) pour le code que vous aurez développé, 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 votre programme.

   (b) `make mesures` : Commande qui permet d'exécuter votre programme 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.

   Remise électronique : Vous devrez remettre une version électronique de votre code source en exécutant la commande suivante sur la machine `zeta` :

   `$ oto rendre_tp tremblay_gu INF7235` <*code(s)Permanent(s)*> <*repertoire*>

   Vous pouvez aussi me remettre un unique *fichier d'archives* contenant l'ensemble des fichiers pour votre code source ainsi que les programmes tests :

   `$ oto rendre_tp tremblay_gu INF7235` <*code(s)Permanent(s)*> <*archives*>*.tar.gz*

   OU

   `$ oto rendre_tp tremblay_gu INF7235` <*code(s)Permanent(s)*> <*archives*>*.zip*

   Note : Si vous êtes en équipe de deux personnes, alors les deux codes permanents doivent être séparés par une «,», par exemple :

   `$ oto rendre_tp tremblay_gu INF7235 ABCD11111111,DEFG22222222` <*archives*>

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

   `$ oto confirmer_remise tremblay_gu INF7235` <*code(s)Permanent(s)*>

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.

   Exemple, pour un programme de tri : On pourrait vérifier que le résultat obtenu est correctement trié — avec des assertions sur l'ordre relatif correct des éléments adjacents — et, encore mieux, vérifier que le résultat est bien une permutation des données initiales. On pourrait aussi vérifier que les résultats produits par les versions parallèles sont identiques entre eux, et identiques à ceux produits par une version séquentielle — **ceci est, assurément, le minimum que vous devriez faire!**

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

Critères de correction

Lors de la remise de votre travail, vous devez utiliser la page de présentation et **la grille de correction** disponible à la page 6.

Note : Cette grille s'applique pour le 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.

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 mercredi, 23 avril, 16h00
Ne pas mettre dans une enveloppe

Nom	
Prénom	
Code permanent	
Courriel	
Nom	
Prénom	
Code permanent	
Courriel	

Description du problème, des solutions choisies et justification, 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
Description de la stratégie de tests, 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		5 pts
Total		55 pts

Note globale :	/ 10