Table des matières

PCP – Travail pratique 01

Ce travail pratique vous amènera à appliquer l'un des schémas de conception de parallélisme les plus connus, soit l'approche *Map/ Reduce*[1].

Illustration – exemple canonique

L'exemple canonique de l'approche *Map/ Reduce* (*exemple qui ne correspond pas au travail demandé ici*) consiste à comptabiliser le nombre de mots dans un fichier, et ce pour un grand nombre de fichiers :

- l'étape *Map* est alors de prendre l'ensemble des fichiers sur lesquels il faut opérer, de séparer cet ensemble en deux sous-ensembles confiés à deux unités de traitement qui opéreront en parallèle (lorsque la cardinalité de l'ensemble à traiter tombe sous un certain seuil, la tâche de fera séquentiellement);
- l'étape *Reduce* consiste quant à elle à recevoir le fruit du calcul des deux tâches concurrentes et de les colliger. Ici, il s'agirait d'évaluer la somme du nombre de mots calculés par les deux tâches concurrentes.

```
compter_mots(S : ensemble fichiers)
    Si |S| < SEUIL_SÉQUENTIEL
        return compter_mots_séquentiel(S)
    S' ← S[début..milieu[
    S'' ← S[milieu..fin[
    // map
    T0 ← thread(compter_mots,S')
    T1 ← thread(compter_mots,S'')
    attendre{T0,T1}
    // reduce
    return résultat(T0)+résultat(T1)
```

En procédant ainsi, dans la mesure où le traitement est effectivement réalisé de manière concurrente, dans la mesure où les deux tâches concurrentes s'exécutent à peu près aussi longtemps l'une que l'autre, et dans la mesure où l'étape de réduction se fait rapidement, cette approche au parallélisme permet d'obtenir des gains d'ensemble intéressants sur le plan de la vitesse d'exécution.

[1] http://h-deb.clg.qc.ca/Liens/Multiprogrammation--Liens.html#algo_map_reduce

Le travail demandé

Nous procéderons à partir d'une variante de l'exemple canonique de cette approche (ce qui ne vous empêche pas d'implémenter l'exemple canonique pour vous faire la main). En effet, **votre travail consistera à compter le nombre d'occurrences de chaque mot correspondant à un certain critère dans un ensemble de fichiers** – un exemple de critère serait « mots d'une longueur d'au moins neuf caractères » ou, dans l'exemple ci-dessous, « pareils sans tenir compte de la casse » – et à retourner un ensemble de paires $\{s, n\}$ où s est un mot pour lequel le critère s'avère et n est le nombre total d'occurrences de s dans l'ensemble de fichiers traités.

Pour nos fins, un mot sera une séquence de caractères délimitée par des blancs, ou par un début ou une fin de chaîne. Ainsi, dans `"Allo, mon cher!"` on trouve trois mots, soit `"Allo,"`, `"mon"` et `"cher!"`.

Imaginons par exemple les trois fichiers ci-dessous :

a.txt	**b.txt**	**c.txt**
Ceci est un test pour qui s'en amuse	*Que faire de ceci, je vous le demande?*	*Et de ceci, qu'en est-il?*

En pratique, vous utiliserez sans doute (du moins, je l'espère!) des fichiers bien plus volumineux, mais pour les besoins de l'explication, ceci suffira.

Si le critère de sélection des mots est « tout mot de longueur entre `4` et `5` inclusivement », alors le résultat du traitement sera la liste de paires visible à droite.

Mot	**Occurrences**
`ceci,`	`2`
`Ceci`	`1`
`amuse`	`1`
`faire`	`1`
`qu'en`	`1`
`pour`	`1`
`s'en`	`1`
`test`	`1`
`vous`	`1`

Vous remarquerez que les mots sont triés en ordre lexicographique. C'est sans doute l'approche la plus efficace dans un tel cas : retrouver un mot parmi ceux déjà rencontrés est alors une tâche de complexité $O(\log_2 n)$ alors que dans une séquence non-triée, retrouver un mot parmi ceux déjà rencontrés serait une tâche de complexité $O(n)$.

En fin de traitement, vous devrez afficher le nombre total de fichiers traités et le nombre total de mots respectant le critère utilisé. Si vous traitez plusieurs fois le même fichier, pour fins de test, chaque traitement d'un fichier donné compte pour un fichier : par exemple, si vous traitez `a.txt`, `b.txt`, `c.txt` et `a.txt`, alors quatre fichiers ont été traités même si `a.txt` apparaît deux fois dans la liste.

Une sortie possible pour l'exécution de votre programme serait la suivante. Ceci est celle de ma propre implémentation, et ne vous est offerte que pour fins d'inspiration :

```
Chargement en memoire du texte des 85 fichiers a traiter... complete.

sequentiel : 2720 fichiers, 3048416 mots, 31661 ms.
Test juste comme ca: nb occurrences du mot "Incopiable" : 1920
Resultats detailles dans "out-sequentiel.txt"

parallele : 2720 fichiers, 3048416 mots, 4539 ms. (seuil sequentiel : 32)
Test juste comme ca : nb occurrences du mot "Incopiable" : 1920
Resultats detailles dans "out-parallele-0.txt"

parallele : 2720 fichiers, 3048416 mots, 5027 ms. (seuil sequentiel : 64)
Test juste comme ca : nb occurrences du mot "Incopiable" : 1920
Resultats detailles dans "out-parallele-1.txt"

parallele : 2720 fichiers, 3048416 mots, 6129 ms. (seuil sequentiel : 128)
Test juste comme ca : nb occurrences du mot "Incopiable" : 1920
Resultats detailles dans "out-parallele-2.txt"

parallele : 2720 fichiers, 3048416 mots, 7504 ms. (seuil sequentiel : 256)
Test juste comme ca : nb occurrences du mot "Incopiable" : 1920
Resultats detailles dans "out-parallele-3.txt"

parallele : 2720 fichiers, 3048416 mots, 8189 ms. (seuil sequentiel : 512)
Test juste comme ca : nb occurrences du mot "Incopiable" : 1920
Resultats detailles dans "out-parallele-4.txt"

parallele : 2720 fichiers, 3048416 mots, 11560 ms. (seuil sequentiel : 1024)
Test juste comme ca : nb occurrences du mot "Incopiable" : 1920
Resultats detailles dans "out-parallele-5.txt"

Algorithme sequentiel: 42.4351% du temps total
Algorithmes paralleles:
        * seuil sequentiel de 32, 6.0838% du temps total
        * seuil sequentiel de 64, 6.73788% du temps total
        * seuil sequentiel de 128, 8.21494% du temps total
        * seuil sequentiel de 256, 10.0579% du temps total
        * seuil sequentiel de 512, 10.976% du temps total
        * seuil sequentiel de 1024, 15.4943% du temps total
```

Avec mon implémentation et mon (petit) ordinateur portatif, un seuil tardif de passage au mode séquentiel (une fois l'échantillon réduit à `32` éléments et moins, disons) est préférable. Qu'en sera-t-il de votre propre implémentation?

Objectif général et objectifs spécifiques

L'objectif général de ce travail est d'implémenter une solution de type *Map/ Reduce* à un problème. Quelques objectifs spécifiques s'y accolent :

- une solution de type *Map/ Reduce* passe à une résolution séquentielle lorsque la taille de l'échantillon à traiter passe sous un certain seuil (par exemple, une fois que le nombre de fichiers à traiter passe sous `64`). Vous devrez tester votre solution parallèle avec plusieurs seuils et évaluer l'impact de la croissance ou de la décroissance de ce seuil sur les performances – étant donné votre façon de faire et vos échantillons, évidemment;
- vous devrez comparer le temps d'exécution d'une solution séquentielle au problème avec le temps d'exécution des diverses solutions parallèles;
- vous devrez vous assurer que les solutions séquentielle et parallèles donnent les mêmes résultats (mêmes mots, même nombre d'occurrences de chaque mot, étant donné les mêmes échantillons en entrée); et
- vous devrez bien sûr vous assurer que l'exécution ne plante pas, et que son résultat soit défini (pas de conditions de course, pas de comportement indéfini[2]).

Si vous êtres curieuse / curieux, comparez ces tests avec une solution découpant l'ensemble des fichiers à traiter de manière à ce que le nombre total de *threads* concurrents corresponde au nombre de cœurs physiques à votre disposition, tel que retourné par un mécanisme tel que la fonction `thread::hardware_concurrency()`. Dans ce cas, pas besoin de subdiviser encore et encore : une seule division correspondant au nombre de cœurs physiques suffira;

Que constatez-vous?

[2] http://h-deb.clg.qc.ca/Sujets/Developpement/Comportement_indefini.html

Consignes générales

Je vous recommande d'écrire initialement le traitement séquentiel, pour déboguer les trucs simples de la mécanique et avoir une solution opérationnelle. Profitez-en pour mesurer les performances à l'exécution de votre programme en mode séquentiel, à titre indicatif. Assurez-vous de pouvoir vérifier le résultat de l'exécution à partir d'un banc d'essai que vous connaissez.

Ensuite, parallélisez le tout selon l'approche exigée. Assurez-vous que les résultats sont les mêmes que dans la démarche séquentielle.

Faites vos tests en mode *Release* (ou équivalent, p. ex. : optimisation `-O2` avec `g++`); des tests de vitesse d'exécution réalisés avec une version de débogage, ça ne vaut rien. Vraiment rien.

Aussi, faites vos tests une première fois, puis refaites-les. Les systèmes d'exploitation contemporains procèdent à un certain nombre d'optimisations importantes sur les données qui ont récemment été chargées en mémoire, alors la vitesse d'une première exécution d'un programme donné est en général nettement inférieure à celle des exécutions subséquentes.

Je suis ouvert aux travaux dans divers langages de programmation, dans la mesure où vous me consultez au préalable si vous sortez du groupe commercial classique (C, C++, Java, C#). J'ai évidemment utilisé C++ pour ma version, que je vous présenterai une fois votre version livrée; nous pourrons échanger sur les techniques employées dans votre code comme dans le mien, critiquer, réfléchir collectivement, *etc*.

Vous pouvez travailler seul(e) ou en équipe de deux ou trois. Pour les équipes de trois, le petit bonus (ci-dessous) sera exigé et ne sera pas un bonus.

Petit bonus : si vous parvenez à gérer les *proprement* exceptions levées dans une sous-tâche à même la tâche l'ayant lancée (ou « plus haut » dans la séquence de lancement de tâches), et si votre approche est documentée, je vous accorderai un petit bonus étant donné que nous n'avons pas encore vu comment le faire (*ergo*, il vous faudra chercher un peu).

Quoi remettre

Je m'attends à un rapport mettant en relief les éléments suivants :

- l'identité et la contribution de chaque membre de l'équipe;
- une description stratégie choisie pour implémenter la partie parallèle (incluant la synchronisation). Cette partie doit inclure le détail des structures de données, idiomes de programmation et schémas de conception que vous aurez choisi d'appliquer pour y arriver (code à l'appui);
- les spécifications clés de l'ordinateur sur lequel les tests ont été faits (mémoire vive, nombre de cœurs ou de processeurs);
- le détail de votre stratégie de mesure pour la version séquentielle, et les métriques obtenues; de même que
- le détail de votre stratégie de mesure pour la version parallèle, et les métriques obtenues.

Bien que toutes les parties soient importantes (n'en négligez aucune!), je m'attarderai plus spécifiquement sur la section indiquant l'implémentation de la version parallèle, de même que sur les stratégies de mesure. La grille de correction sera pondérée en conséquence.

Format de la remise

La remise du travail peut être sous forme papier ou sous forme électronique. Dans le cas d'un document électronique, si vous souhaitez une rétroaction signifiante, je dois être en mesure de l'annoter à l'aide de Microsoft Word 2013 (je n'ai *vraiment* rien contre Open Office, mais ce n'est tout simplement pas ce que j'ai sur mes postes de travail). Il est possible mais un peu déplaisant d'annoter un PDF, alors si vous me livrez un document dans ce format, je limiterai ma rétroaction à quelques remarques par courriel, sans plus.

Si vous souhaitez que je teste votre travail chez moi, tenez compte des spécifications techniques de mon ordinateur[3]. Vous devez aussi vous assurer, par souci d'intégrité, de pouvoir me produire les sources de tous vos modules sur demande et de pouvoir me donner une démonstration sur place, au campus où vous suivez ce cours et un jour où je suis présent, si j'en exprime le désir (si je le fais, je vous donnerai quelques jours d'avis).

Ne me rendez pas la vie trop compliquée je vous prie : il est nécessaire que je puisse installer et désinstaller le tout aussi simplement que possible, que ce soit par un installateur automatisé ou à l'aide de consignes simples et claires. Je corrige énormément de travaux chaque session; il est donc impératif que vous me rendiez la tâche facile à l'installation et que mon ordinateur soit *propre* après la désinstallation.

Tout document remis devra présenter un français de qualité. Le code devra utiliser une police de caractères non proportionnelle (quelque chose comme Courier New ou Lucida Console par exemple) pour l'impression de programmes ou d'extraits de programmes, et indentez à l'aide d'espaces (les tabulations ont une influence qui varie d'un éditeur à l'autre, ce qui complique la présentation du code et sa lisibilité). Ces polices conviennent bien à un tel usage, clarifiant l'indentation appliquée[4]. D'ailleurs, même si vous utilisez des outils de génération automatique de code[5], ce code devra être indenté de manière lisible, sinon il vous sera retourné sans que je ne l'aie lu.

Si vous choisissez de faire une remise en format imprimé, cette remise devra être faite le **lundi 17 juillet 2017 en début de séance**. Si vous préférez livrer une version électronique par courriel à Patrice.Roy@USherbrooke.ca, alors cette remise devra être faite le **lundi 17 juillet 2017 à 17 h**.

[3] PC au processeur Intel à quatre cœurs cadencés à 2,6 GHz, 7,6 Go RAM, *Microsoft Windows* 7, ce qui devrait suffire.

[4] Sur le site de l'enseignant, voir les articles http://h-deb.clg.qc.ca/Sujets/TrucsScouts/Police-pour-code.html et http://h-deb.clg.qc.ca/Sujets/TrucsScouts/Eviter-Tabulations.html pour plus de détails quant aux exigences en lien avec la présentation du code.

[5] Je pense en particulier aux *aficionados* de Delphi ici. http://h-deb.clg.qc.ca/Sujets/Developpement/Pratique-programmation.html#indentation pour des idées. L'important est d'être conséquent à l'interne dans un programme donné, et de viser la lisibilité.