UFR IM²AG

UNIVERSITÉ Grenoble Alpes

Master M1 Informatique, CSE
TD & TP Evaluation de performances
Année 2018-2019

1 Présentation

Dans cette séance nous allons considérer les performances à l'exécution d'un programme multi-threadé. Nous allons nous intéresser à l'accélération du programme en fonction de la taille des données traitées et du nombre de threads.

L'ensemble des manipulations en TP se fera en utilisant un ensemble de programmes permettant de trier les éléments d'un vecteur par ordre croissant. Ces programmes se trouvent dans l'archive `tri-1.0.tar.gz`, que vous pouvez décompresser et extraire à l'aide des deux commandes suivantes :

```
gzip -d tri-1.0.tar.gz
tar xvf tri-1.0.tar
```

Les programmes contenus dans cette archive sont décrit dans le fichier `README` qu'elle contient, l'ensemble se compile de la manière suivante :

```
./configure
make
```

La cible `test` du Makefile produit un ensemble d'exécutions qui illustrent la manière dont s'utilisent les programmes fournis.

2 Utilisation de `gettimeofday`

L'appel `gettimeofday` permet de récupérer une information sur l'heure courante dans le système. Son prototype est le suivant :

```
int gettimeofday(struct timeval *restrict tp, void *restrict tzp);
```

Cet appel renvoie son résultat dans la structure `timeval` fournie sous la forme du temps écoulé en microsecondes depuis le premier janvier 1970. Le second argument n'a pas d'importance dans le contexte qui nous intéresse et peut être fixé à NULL. Pour mesurer le temps d'exécution d'un programme ou d'une partie d'un programme, il est donc possible de procéder de la manière suivante :

```
temps1 = mesure_temps...
traitement...
temps2 = mesure_temps...
printf("Le temps de traitement est :...", temps2-temps1);
```

Questions :

1. D'après vous, en reprenant l'exemple de la recherche d'un élément dans un vecteur non trié, à quels endroits du programme est-il judicieux de placer les appels à `gettimeofday` afin de comparer les versions séquentielle et parallèle du programme ? En particulier, doit-on inclure la création de threads et la synchronisation finale dans la prise de mesure ?
2. Pensez vous que le système, en particulier l'ordonnanceur, et les autres utilisateurs aient une influence sur votre programme ? Justifiez votre réponse. Vérifiez votre réponse expérimentalement en exécutant plusieurs fois et pour plusieurs tailles de données l'un des programmes fournis.

3 Utilisation de `getrusage`

L'appel système `getrusage` a le prototype suivant :

```
int getrusage(int who, struct rusage *r_usage);
```

Appeler cette fonction avec pour premier argument `RUSAGE_SELF` permet de récupérer de nombreuses informations concernant l'exécution du processus en cours dans la structure dont l'adresse est fournie en second argument. Dans le cas d'un programme multithreadé, ces informations seront le cumul pour l'ensemble des threads du processus. Parmi les informations intéressantes retournées par cet appel système, nous trouvons le temps cumulé durant lequel un processeur a été attribué au processus en mode utilisateur ainsi que le temps cumulé durant lequel un processeur a été attribué au processus en mode superviseur. `getrusage` s'utilise de manière analogue à `gettimeofday`.

Questions :

1. Pensez vous que le nombre de threads d'un programme ait une influence sur le temps reporté par `getrusage` ? Et le nombre de processeurs (ou cœurs) de la machine ? Effectuez des expériences avec les programmes fournis pour vérifier vos réponses.
2. D'après vous, quelles sont les différences entre l'utilisation de `gettimeofday` et `getrusage` pour la mesure du temps d'exécution d'un algorithme parallèle ? En particulier, le temps total reporté avec `getrusage` est-il inférieur, égal ou supérieur à celui reporté avec `gettimeofday` ? Dans quels cas ?

4 Performance d'un programme

L'utilisation de plusieurs threads pour effectuer un calcul sur les éléments d'un vecteur (dans notre cas un tri) a pour but principal d'aller plus vite. Il est donc important d'être capable que quantifier le gain en performances apporté par une exécution parallèle. Dans toute la suite, vous pourrez remarquer que les mesures de performance se font à partir de temps d'exécution. Elles sont donc fortement liées à une machine donnée. Les caractéristiques de la machine cible, en particulier son nombre de processeurs, seront par conséquent une donnée importante faisant partie des conditions expérimentales. A l'UFR IM^{2}AG, vous pourrez mener vos expérience soit :

- en local, sur le poste sur lequel vous êtes connecté. Sous Linux, vous pourrez vérifier le nombre de processeurs dont vous disposez en exécutant la commande : `cat /proc/cpuinfo`
- sur mandelbrot, qui possède 4 processeurs
- sur imasrv-compil, qui correspond à une connection sur une des imablades de l'UFR qui possèdent toutes 2 processeurs à 4 coeurs

La première métrique couramment utilisée dans le contexte du calcul parallèle est l'accélération. L'accélération d'un programme parallèle est le rapport entre le temps d'exécution de la meilleure version séquentielle résolvant le même problème et le temps d'exécution de la version parallèle. Cette mesure nous donne donc la performance absolue d'un algorithme donné sur une machine donnée. La seconde métrique également très répandue est l'efficacité. Elle est égale au rapport entre l'accélération et le nombre de processeur de la machine utilisés par le programme. Elle exprime donc une performance relative aux ressources fournies.

Questions :

1. D'après vous, quels facteurs influencent l'accélération et l'efficacité d'un programme parallèle :
   - les algorithmes utilisés ?
   - le nombre de threads utilisés ?
   - la bibliothèque de threads utilisée ?
   - le système d'exploitation ?
   - le nombre de processeurs de la machine ?
   - la vitesse des processeurs ?
   - l'architecture mémoire de la machine ?

— les utilisateurs connectés sur la machine ?
— autre ?

Dans chacun des cas envisagés, expliquer dans quelles circonstances l'accélération ou l'efficacité est augmentée, diminuée et pourquoi. Verifiez vos hypothèses à l'aide des programmes de tri fournis.

2. Nous allons maintenant établir un plan d'expérimentation afin de mesurer la performance de nos algorithmes de tri. Vous commencerez par préciser vos conditions expérimentales, c'est-à-dire les conditions d'exécution sur lequelles vous n'avez pas d'influence (modèles des machines, charge respective, ...). Ensuite vous tenterez d'évaluer l'accélération et l'efficacité obtenue par chacun des algorithmes en fonction de leurs paramètres. En supposant les différents paramètres indépendants (ce qui n'est pas toujours le cas) il faudra donc mesurer la performance de nos programmes en faisant varier un paramètre à la fois. Vous pourrez, par exemple, utiliser le plan d'expérimentation suivant :
   — à nombre de threads fixé, mesurez l'accélération et l'efficacité de chaque programme en fonction de la taille du vecteur utilisé (de 1000 à 1000000 éléments).
   — à taille de vecteur fixée, mesurez l'accélération et l'efficacité de chaque programme en fonction du nombre de threads (de 1 à 16).

   Pour chacun des points que vous mesurerez, il faudra autant que possible eliminer les incertitudes de mesure. Pour cela, vous répéterez l'expérience au moins 30 fois et ne retiendrez que la moyenne. Vous remarquerez que pour effectuer l'ensemble de vos mesures, il vous sera profitable d'ecrire un ou plusiers scripts lançant toutes les exécutions necessaires.

   Une fois l'ensemble de vos mesures effectué, tracez les courbes d'accelération et d'efficacité correspondantes. Que pensez vous de la pertinence de vos mesures ? Comment la garantir ?

5 Poussez les expériences plus loin

Il reste encore quelques éléments non évalués dans le plan expérimental proposé dans la partie précédente. Dans cette partie, nous vous proposons de pousser les expérimentations plus loin.

Questions :

1. Mesurez l'influence des valeurs du vecteur aléatoire généré sur les performances du tri
2. Le temps moyen mesuré jusqu'alors n'apporte qu'une information partielle sur les expériences. En particulier, la variabilité des mesures est une information essentielle permettant de quantifier la confiance que nous pouvons avoir dans les résultat. Evaluez la variabilité de vos mesures précédentes en calculant pour chaque point son intervalle de confiance à 95%. Completez vos courbes en y faisant figurer ces intervalles.

6 Autres algorithmes

Nous allons maintenant implémenter de nouveaux algorithmes de calcul sur des vecteurs en utilisant le squelette fourni. Etudiez les fichiers `appels_sequentiels.h` et `tri.c` qui contiennent respectivement la description de l'interface utilisée par un algorithme de calcul sur un vecteur et son utilisation dans le cas du tri. Etudiez également le fichier `Makefile.am` qui décrit la manière dont les exécutables sont fabriqués à partir des sources.

Questions :

1. Implémentez la recherche d'un élément (donné en argument sur la ligne de commande) dans le vecteur lu en entrée. Mesurez ses performances.
2. Implémentez un algorithme qui compte le nombre de nombres premiers présent dans le vecteur lu en entrée. Mesurez ses perofrmances.