ASR7 Programmation Concurrente

—

Sujet bonus: produit matrice-vecteur

Sylvain Brandel, Yves Caniou, Guillaume Damiand, Meriem Ghali,
Laurent Lefèvre, Thibaut Modrzyk, Grégoire Pichon, Alec Sadler,
Florence Zara, Jerry Lacmou Zeutouo

Printemps 2024

I Disclaimer

L'objectif de ce sujet est de vous familiariser avec une thématique que nous avons peu eu le temps d'aborder dans le cours et les autres TP : l'efficacité parallèle.

Il est volontairement assez peu guidé, pour que vous expérimentiez par vous même différentes stratégies de parallélisation. Selon votre enthousiasme dans le sujet, il pourra être complété / complexifié. N'hésitez pas à en discuter sur le chat, où nous pourrons vous aider à mieux comprendre certains phénonmènes !

Ce TP est présenté comme un challenge, vous devez obtenir les meilleures performances possibles et de préférence meilleures que vos collègues. Soyez fair-play, l'idée est de vous amuser un peu mais pas de vous mettre en compétition entre vous ! Le TP n'est pas noté.

II Présentation du problème

On s'intéresse dans ce sujet à la parallélisation d'un produit matrice-vecteur, une opération de base en algèbre linéaire, utile à de nombreuses applications scientifiques et industrielles.

Le fichier se compile avec la ligne de commande suivante :

```
g++ -g -Wall -pthread -std=c++11 matrix_vector_product.cpp -o matrix_vector_product
```

Si vous exécutez votre programme avec 4 threads et une matrice de taille 3000 avec la commande `matrix_vector_product 4 3000` vous devriez obtenir une sortie similaire à :

```
Using 4 threads for a matrix of size 3000
Initialization ... done.
Sequential computation ... done.
Parallel computation ... done.
Sequential: 0.030517s, Parallel: 0.031092s, Speedup: 0.981506, Max speedup: 4
Norm of ||c2-c1||: 0, Norm / n should be very small: 0
```

Vous pouvez observer que le temps séquentiel et le temps parallèle sont très proches. Si vous regardez en détail le code, vous verrez que 4 threads sont créés et appellent la fonction `fct_v0`. Cette fonction appelle le code séquentiel pour le premier thread (d'indice 0 donc) et

ne fait rien pour les autres threads. On a donc une implémentation parallèle qui est identique à celle séquentielle. Le speedup, donc le ratio entre le temps séquentiel et le temps parallèle est autour de 1 : on n'accélère pas le code !

L'objectif est d'accélérer notre produit matrice-vecteur. Pour cela, à vous de jouer ! Testez différentes solutions avant de converger vers une implémentation efficace. Attention, avec l'approche la plus naïve, vous aurez peut-être un speedup inférieur à 1 : votre code parallèle prendra plus de temps que le code séquentiel ! Cela peut être du aux accès mémoire. Comme discuté en cours, afin de tirer profit des architectures (mémoire, caches etc...) il faut favoriser la localité spatiale et temporelle. En particulier, on pourrait penser que les imbrications `for (i = ...) for (j = ...) {...}` et `for (j = ...) for (i = ...) {...}` sont équivalentes, mais ce n'est pas du tout le cas en terme de performance !

Vous devez implémenter la fonction `fct_v1` (voire `fct_v2` pour une version plus efficace). Pour exécuter votre code avec cette nouvelle fonction, vous devrez utiliser `./matrix_vector_product 4 3000 fct_v1`. Pour bien mesurer l'efficacité de votre algorithme, il faudra utiliser différentes tailles de matrices, qui auront des effets non négligeables sur les caches.

III Rendu

Vous avez une case `speedup_product` à compléter sur TOMUSS avec le speedup obtenu en utilisant votre algorithme le plus efficace sur une matrice de taille $n = 20000$ en utilisation 4 coeurs. L'objectif est d'atteindre une valeur la plus proche possible de 4, en supposant que le nombre de coeurs de votre machine est supérieur ou égal à cette valeur. Vous devez rentrer votre speedup sous la forme d'une note comprise entre 0 et 4, en conservant deux chiffres après la virgule. Vous pourrez mesurer l'efficacité de votre approche par rapport à celle des autres en regardant le classement sur votre suivi TOMUSS sur cette note. L'idéal serait de réaliser vos mesures de temps sur les machines du Nautibus, afin d'avoir des résultats comparables entre vous. Vous devez également déposer votre fichier dans la case `code_product`. De cette manière, nous pourrons comparer les résultats de votre implémentation dans les mêmes conditions (machine, version du compilateur etc...).

Il vous est demandé d'utiliser la ligne de compilation fournie dans ce document et dans le fichier .cpp. Vous pourriez utiliser des options d'optimisation de g++ (-03 par exemple), mais les comportements des différentes versions que vous proposerez seront plus complexes à analyser. Vous ne devez donc pas le faire.

Afin de mieux illustrer le comportement de votre approche, vous pourrez tracer l'évolution du speedup en fonction de la taille de la matrice (x=n, y=speedup = f(n)), en utilisant différents nombres de threads. Pour cela, vous pouvez utiliser la bibliothèque `matplotlib` de `python`.