Exemple de rapport attendu

Juvigny Xavier

Février 2023

1 Présentation de l'ordinateur sur lequel le TP a été effectué

L'ordinateur sur lequel le TP a été réalisé est un intel celeron possédant quatre cœurs physiques de calcul sous Linux. Voici la sortie que nous a donné lscpu :

```
Architecture :                                  x86_64
  Mode(s) opératoire(s) des processeurs :       32-bit, 64-bit
  Address sizes:                                39 bits physical, 48 bits virtual
Processeur(s) :                                 4
  Liste de processeur(s) en ligne :             0-3
Identifiant constructeur :                      GenuineIntel
  Nom de modèle :                               Intel(R) Celeron(R) N4120 CPU @ 1.10GHz
    Famille de processeur :                     6
    Modèle :                                    122
    Thread(s) par cœur :                        1
    Cœur(s) par socket :                        4
    Socket(s) :                                 1
    ...
Caches (sum of all):
  L1d:                                          96 KiB (4 instances)
  L1i:                                          128 KiB (4 instances)
  L2:                                           4 MiB (1 instance)
```

Notons que nous n'avons pas de cache L3 sur cet ordinateur.

2 Produit matrice-matrice

2.1 Tests sur la permutation des boucles

Dans un premier temps, on s'est contenté de permuter les boucles pour le produit matrice-vecteur et nous avons regarder pour diverses dimensions le temps pris pour calculer le produit matrice-matrice. Voici les résultats obtenus dans le tableau suivant (en MFlops (secondes)) :

Ordre des boucles	dim 1023	dim 1024	dim 1025
ijk	14.06s (152.24 MFlops)	20.52s (104.7 MFlops)	13.42s (160.45 MFlops)
ikj	26.32s (81.34 MFlops)	43.9s (48.9 MFlops)	27.19s (79.2 MFlops)
jik	13.15s (162.8 MFlops)	20.05s (107.1 MFlops)	12.91s (166.8 MFlops)
jki	2.22s (971.6 MFlops)	2.23s (961.4 MFlops)	2.226s (967 MFlops)
kij	26.4s (81.1 MFlops)	53.74s (40 MFlops)	27s (79.6 MFlops)
kji	2.6s (820.7 MFlops)	2.55s (840 MFlops)	2.63s (816 MFlops)

Observation :

Il est clair que les meilleures performances sont obtenues lorsque la boucle en i est la plus interne. De plus, dans ce cas, la variation du temps de calcul en fonction de la dimension de la matrice devient insignifiante. Les meilleures performances étant obtenues avec les boucles dans l'ordre jki.

Interprétation :

On sait que la matrice est stockée par colonne, et donc que le premier indice est celui qui en variant permet d'accéder à des données contigües en mémoire. Puisque le produit matrice vecteur fait apparaître dans la boucle la plus interne le calcul $C_{ij} \leftarrow C_{ij} + A_{ik}.B_{kj}$, on voit qu'en mettant la boucle en i comme la plus interne, cela permet d'accéder aux coefficients de C et de A en contigü dans la mémoire, tandis que pour B, on conserve la même valeur durant toute la boucle interne. En mettant k en boucle du milieu, cela permet également d'accèder en contigü aux cœfficients de B.

En revanche, lorsque la boucle en j est la plus interne, on obtient les plus mauvais performances puisque dans ce cas, l'accès aux cœfficients de C et de B se fait avec des sauts mémoires dont la distance est égale à la dimension de la matrice. De plus, si la taille de la matrice est une puissance de deux, on tombera plus souvent sur une adresse mémoire de la mémoire cache qui contient déjà une valeur lue ou écrite récemment, ce qui explique cette sensibilité à la dimension de la matrice (ce qui n'est pas le cas lorsque la boucle en i est la plus interne, puisque dans ce cas on lit les cœfficients en contigü).

Conclusion :

On voit qu'il est important de comprendre comment marche la mémoire cache afin d'obtenir une bonne optimisation de nos codes. Dans le cas du produit matrice-matrice, on voit que les gains obtenus sont très significatifs et nous invite à devoir réfléchir dans les sections critiques en temps de nos codes à réfléchir sur la façon dont on accèdera aux données (et à la représentation des donnée en mémoire).

2.2 Parallélisation de la boucle externe en OpenMP

Nous allons nous intéresser à la parallélisation de la boucle externe dans le cas le plus favorable et mesurer l'accélération obtenue à l'aide d'une matrice de dimension 2048

Nombre de cœurs utilisés	Temps	Accélération	Efficacité
1	18.2s	1	100%
2	10s1s	1.80	90%
4	6s1	3	75%

Remarque : il est inutile de prendre plus de cœur sur l'ordinateur que nous avons utilisé pour le TP, puisque ce dernier ne contient que quatre cœurs physiques de calcul.

Nous allons maintenant nous intéresser à conserver une quantité de travail constant par cœur de calcul. Pour cela, nous allons prendre une matrice de dimension 1290 pour un calcul sur un cœur, 1625 pour le calcul sur deux cœur et 2048 pour le calcul sur quatre cœurs. Ainsi, sur un cœur, on devra effectuer 2 146 689 000 $(+, \times)$, sur deux cœurs 2 145 507 812 $(+, \times)$ par cœur de calcul et sur quatre cœurs, 2 147 483 648 $(+, \times)$ par cœur de calcul.

Voici l'accélération pour chaque cas cité au-dessus :

Nombre de cœurs utilisés	Accélération	Efficacité
1	1	100%
2	1.8	90%
4	3	75%

On remarque que l'accélération est la même dans le premier cas, défavorable (qui suit

la loi de Moore) et pour le second, favorable (qui suit la loi de Gustavson). La dégradation des performances est donc sûrment dû d'une part à l'accès au cache commun des cœurs de calcul et à une légère saturation de la bande passante mémoire avec l'augmentation de nombre de cœurs, mais cela n'est pas très clair du fait de la lenteur de traitement des cœurs de calcul employés (la saturation de la bande passante mémoire devrait être plus marquée avec des cœurs de calcul plus performants).

2.3 Approche par bloc

Nous allons tenter une approche par bloc afin d'améliorer les performances. Le gain espéré ne devrait pas être très grand au vu de la remarque faite dans la sous-section précédente.

Nous allons dans un premier temps voir l'influence de la taille des blocs sur les performances obtenues en séquentiel pour une matrice de dimension 2048

Taille du bloc	Temps de calcul
32	14s
64	15s
128	14s
256	14s

Il est clair que sur l'ordinateur employé, la dimension des blocs influt peu sur la performance finale, sûrement à cause d'une bande passante mémoire peu saturée.

2.4 Parallélisation de l'approche par bloc

Nous allons regarder maintenant la performance obtenu en fonction de la taille des blocs lorsqu'on parallèlise l'approche par bloc avec OpenMP (quatre threads):

Taille du bloc	Temps de calcul	Accélération	Efficacité
32	3s83	3.67	92%
64	4s25	3.56	89%
128	3s72	3.78	94%
256	4s13	3.48	87%

Avec la parallélisation, il devient clair que la meilleure option est de prendre une taille de bloc de 128. Cela permet d'avoir une granularité suffisante pour chaque cœur de calcul mais aussi d'avoir suffisamment de produit bloc-bloc à paralléliser.

Le gain en performance au total est gigantesque, puisqu'on passe de 100 MFlops environ à 2.2 GFlops, soit un temps de calcul divisé par vingt-deux !

Cependant, si on teste le produit matrice-matrice proposé par une bibliothèque BLAS optimisée (ici OpenBlas), on arrive à un temps de calcul pour le produit de matrices de dimension 2048 de 0.75 secondes, soit une puissance de calcul de plus de 22 GFlops, soit un temps de calcul divisé par plus de 220 !

En conclusion, il est toujours préférable d'utiliser des bibliothèques existantes connues pour leurs performances plutôt que d'essayer d'optimiser par soi-même des algorithmes connus (sauf si on a une idée originale qui peut battre les algorithmes connus...)