Cours 3 - C

Imad Kissami

Université Mohammed VI Polytechnique - Licence S3

18 Novembre 2020

Quelques définitions

FLOPS:

Flops correspond aux opérations en virgule flottante par seconde. Il est exprimé en FLOPS ou flop/s

La latence de la mémoire (memory latency):

La latence de la mémoire est le temps entre le lancement d'une requête pour un octet ou un mot en mémoire jusqu'à ce qu'il soit récupéré par le CPU.

La bande passante mémoire (memory bandwidth):

La bande passante mémoire ou débit de la mémoire est la vitesse à laquelle les données peuvent être (lues à partir de) ou (stockées dans) une mémoire à semi-conducteurs par le processeur. Il est exprimé en unités d'octets/seconde.

Intensité de calcul : (Computational intensity)

Définition

Les algorithmes ont deux coûts (mesurés en temps ou en énergie):

- Arithmétique (FLOPS)
- Communication: transfert de données entre :
  - niveaux d'une hiérarchie mémoire (cas séquentiel)
  - processeurs sur un réseau (cas parallèle)

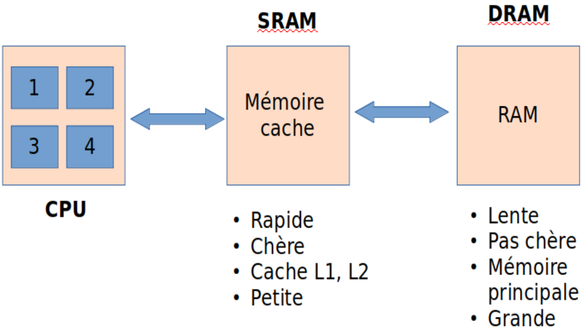
Intensité de calcul

C'est le rapport entre la complexité arithmétique (ou coût) et la complexité de la mémoire (coût).

- Le coût des opérations arithmétiques (par exemple add et mul en virgule flottante) est lié à la fréquence,
- Le coût des opérations de mémoire est le coût du déplacement des données

N.B : Puisque déplacer un mot de données est beaucoup plus lent que de faire une opération dessus, nous voulons utiliser des algorithmes avec une intensité de calcul élevée.

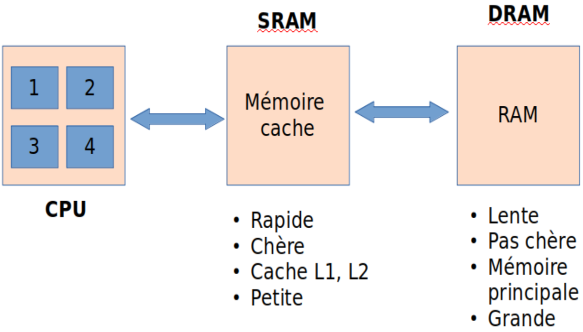
Hiérarchie de la mémoire



Tailles possibles pour les différentes mémoires :

- RAM $\sim$ 4 GB - 128 GB (Taille plus grande dans les serveurs)
- L3 $\sim$ 4 MB - 50 MB
- L2 $\sim$ 256 KB - 8 MB
- Contient les données susceptibles d'être accédées par le CPU
- L1 $\sim$ 256 KB

Hiérarchie de la mémoire



- Cache Hit: si le CPU est capable de trouver la donnée dans L1
- Cache Miss : si le CPU n'est pas capable de trouver la donnée dans L1-L2-L3 et doit la recevoir de la RAM.

Le temps d'exécution

Le temps d'exécution d'un algorithme est la somme de :

- N_flops * temps_par_flop
- N_mots / bande passante
- N_messages * latence

Il est important de noter que : temps_par_flop << 1 / bande passante << latence

Éviter la communication dans les algorithmes permet une meilleure accélération.

Quelques exemples

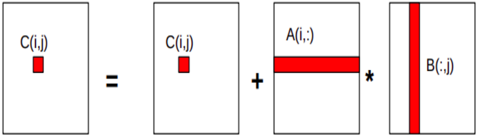
- Jusqu'à 12 fois plus rapide pour matmul 2.5D sur IBM BG/P 64K coeurs.
- Jusqu'à 3 fois plus rapide pour les contractions tenseur sur le noyau 2K Cray XE/6
- Jusqu'à 6,2 fois plus rapide pour All-Pairs-Shortest-Path sur 24K core Cray CE6
- Jusqu'à 2,1 fois plus rapide pour 2.5D LU sur 64K coeurs IBM BG/P
- Jusqu'à 11,8 fois plus rapide pour un N-body direct sur un noyau IBM BG / P 32K
- Jusqu'à 13 fois plus rapide pour Tall Skinny QR sur le GPU Tesla C2050 Fermi NVIDIA

Optimisation de la mémoire : modèle simple à 2 niveaux

- On suppose 2 niveaux de mémoire pour plus de simplicité: rapide et lent. Vous pouvez penser que c'est la RAM et un niveau de cache.
- Initialement, les données sont stockées dans une mémoire lente. Nous définirons ce qui suit.
  - m = nombre d'éléments de mémoire déplacés entre la mémoire lente et rapide
  - tm = temps par opération de la mémoire lente
  - f = nombre d'opérations arithmétiques
  - tf = temps par opération arithmétique
  - q = f/m (nombre moyen de flops par accès lent aux éléments)
- Le temps minimum possible = $f * tf$, lorsque toutes les données sont dans la mémoire rapide
- Le temps réel = $f * tf + m * tm = f * tf * (1 + (tm/tf) * (1/q))$
- Quand q est très grand cela signifie que le temps est plus proche du minimum $f * tf$

Multiplication matrice matrice : version naive

```
for (int i = 0; i < n; i++)
  for (int k = 0; k < n ; k++)
        for (int j = 0; j < n ; j++)
                C[i][j] = C[i][j] + A[i][k] * B[k][j];
```



Multiplication matrice matrice : version naive

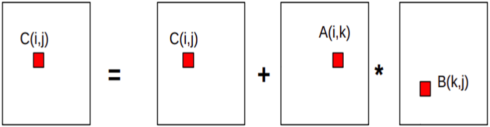
```
for (int i = 0; i < n; i++)
    for (int k = 0; k < n ; k++)
        for (int j = 0; j < n ; j++)
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

- coût arithmétique : $2n^3$ opérations arithmétiques
  - n^3 addition + n^3 multiplication
- coût de mémoire : : $n^3 + n^2 + 2n^2$
  - n^3 : lire chaque colonne de B n fois (une colonne a n éléments et à travers i et j elle accède à n^2 fois)
  - n^2 : lire chaque ligne de A n fois (une ligne a n éléments et à travers i, elle accède n fois)
  - $2n^2$: lire et écrire chaque élément de C une fois
- intensité de calcul : $2n^3/(n^3 + n^2 + 2n^2) = 2$ (n est très grand)

Multiplication matrice matrice : version en bloc

Considérons A, B, C comme N par N matrices de b par b sous-blocs où $b = n/N$ est appelé la taille de bloc.

```
for (int i = 0; i < N; i++)
  for (int k = 0; k < N ; k++)
    for (int j = 0; j < N ; j++)
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
      (multiplication matrice matrice par bloc)
```



Multiplication matrice matrice : version en bloc

Considérons A, B, C comme N par N matrices de b par b sous-blocs où b = n/N est appelé la taille de bloc.

```
for (int i = 0; i < N; i++)
  for (int k = 0; k < N ; k++)
    for (int j = 0; j < N ; j++)
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

- coût arithmétique : $2n^3$ opérations arithmétiques
  - n^3 addition + n^3 multiplication
- coût de mémoire : : $N * n^2 + N * n^2 + 2n^2 = (2N + 2) * n^2$
  - $N * n^2$: lire chaque bloc de B N^3 fois $(N^3 * n/N * n/N)$
  - $N * n^2$: lire chaque bloc de A N^3 fois
  - $2n^2$: lire et écrire chaque élément de C une fois $(= 2 * N^2 * (n/N)2 = 2$ * nombre de fois * élems par bloc))
- intensité de calcul : $2n^3/((2N + 2) * n^2) = n/N = b$ (n est très grand)

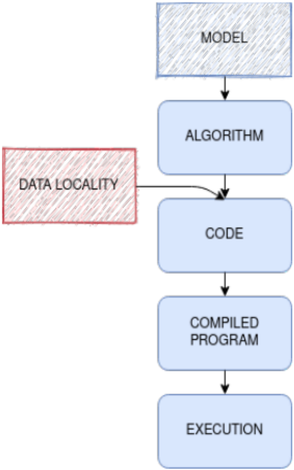
Multiplication matrice matrice : version en bloc

Limite de la taille du bloc :

- Nous pouvons donc améliorer les performances en augmentant la taille du bloc b
- Peut être beaucoup plus rapide que la multiplication naive matrice-matrice (q = 2)
- Cependant, nous ne pouvons pas rendre les tailles de matrice arbitrairement grandes car les trois blocs doivent tenir dans la mémoire.
- Si la mémoire rapide a la taille Mfast
  - $3b^2 <= Mfast$
  - q $= b <= (Mfast/3)^{1/2}$

Localité de données

- La localité des données est souvent le problème le plus important à résoudre pour améliorer les performances.
- Nous avons vu que nous avons 4 niveaux de mémoire
- Où dans cette hiérarchie le processeur trouvera-t-il réellement les données nécessaires à un moment donné?
- Nous pouvons gagner une accélération de $10 - 100$ encore plus élevée par quelques manipulations simples ou plus complexes
- Dans l'exemple précédent, nous avons vu qu'il est possible d'augmenter l'intensité de calcul en réécrivant la multiplication $m * m$ à l'aide d'une version en bloc
- Comprenons donc comment les choses sont réellement gérées



Localité de données

Pénalité du **stride**

Comment accédons-nous aux données?

- Non seulement il existe différentes hiérarchies de mémoires, chacune avec un coût mémoire spécifique
- Nous devons également réfléchir à la manière dont nous accédons aux données
- Nous devons toujours organiser les données de manière à ce que les éléments soient accessibles avec unité (1) stride
- L'exemple suivant vous convaincra que la sanction de ne pas le faire peut être assez sévère

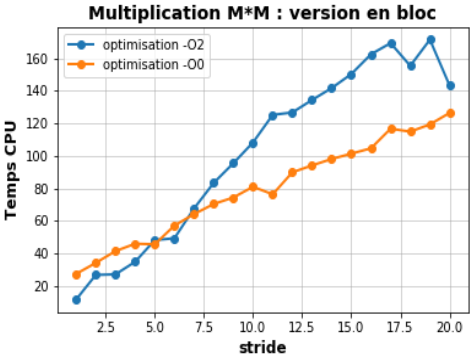
```c
for(int i = 0; i < N*i_stride; i+=i_stride)
  mean = mean + a[i];
```

- Nous compilons le code C ci-dessus en désactivant toute l'optimisation et la vectorisation (-O0) et nous l'exécutons pour différents strides.
- On fait la même chose, avec (-O2) qui active certaines optimisations

Localité de données

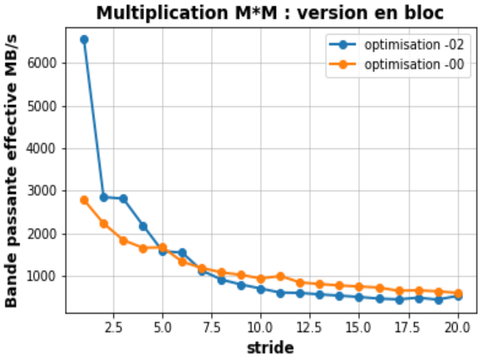
Pénalité du Stride: Temps CPU

- (Ubuntu 7.5.0-3ubuntu1 18.04) 7.5.0
- Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz



Pénalité du Stride:: Bande passante effective

- (Ubuntu 7.5.0-3ubuntu1 18.04) 7.5.0
- Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz



Localité de données

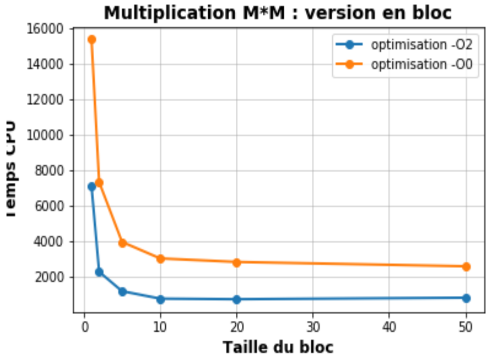
Les tableaux à haute dimension

- Les tableaux à haute dimension sont stockés sous la forme d'une séquence contiguë d'éléments
  - Fortran utilise l'ordre Colonne-Majeure
  - C utilise l'ordre des Ligne-Majeure
- mxm dans C ($N = 1000$)
  - Version naïve : temps CPU 1929.921 (ms)
  - Version de la transposée : temps CPU 583.918 (ms)

Localité de données

Multiplication mxm version en bloc : Temps CPU

- (Ubuntu 7.5.0-3ubuntu1 18.04) 7.5.0
- Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz



Multiplication mxm version en bloc : Bande passante effective

- (Ubuntu 7.5.0-3ubuntu1 18.04) 7.5.0
- Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz