Maïssa KHAMIS
Chun HUYNH
10/03/19

TD-TP n°5 : Simulation d'un cache de données

3. Produit scalaire

On observe que le taux d'erreur varie en fonction de la taille du vecteur, de l'associativité du cache et de la taille du cache.

On sait que l'opération du produit scalaire se fait par l'accès de x[i] et y[i] et que la taille d'un double est de 8 octets.

Prenons par exemple un vecteur de taille N = 64 :

- Si une ligne du cache fait 16 octets, alors une ligne peut stocker 16/8 = 2 double.
  Le taux d'erreur est aussi dû au nombre de défauts de cache, ainsi dans notre cas on a un défaut de cache 1 fois sur 2 car la ligne stocke x[i] mais aussi x[i+1] pour la prochaine opération, ce défaut de cache est l'absence de x[i+2] dans le cache lorsque l'on en aura besoin (1 défaut de cache sur 2 accès mémoire). D'où un miss rate de 0.5.

- De même pour une ligne de 32 octets, qui pourra stocker 32/8 = 4 double.
  On aura dans ce cas 1 défaut de cache sur 4 accès mémoire, d'où un miss rate de 0.25.

- Pareil pour une ligne de 64 octets, qui pourra stocker 64/4 = 8 double.
  On aura dans ce cas 1 défaut de cache sur 8 accès mémoire, d'où un miss rate de 0.125.

Pour un vecteur de taille N = 512 :

Ici, on peut observer approximativement les mêmes résultats que pour un vecteur de taille N = 64, à la seule différence la configuration à 1 ligne par set avec une taille du cache de 4096 octets. Le problème, dans ce cas est que la taille du cache est multiple de 8 (512*8 = 4096), ainsi les éléments des vecteurs x et y sont attribués à la même ligne du cache. Dans une telle configuration on se retrouve avec un défaut de cache par instruction, du fait que l'on soit forcé à réécrire sur la même ligne que celle réservé à x[i] pour y[i] lors de l'opération x[i] * y[i] et ainsi le cache n'a plus la valeur de x[i] qui a été remplacée par y[i].

Pour un vecteur de taille N = 1000 :

Ici, on retrouve approximativement les mêmes résultats que pour un vecteur de taille N = 64.

Pour un vecteur de taille N = 1024 :

On a le même problème que pour un vecteur de taille N = 512. En effet la taille du cache est

multiple de 8 (512*8 = 4096 et 1024*8 = 8192 car ici un vecteur a une taille de 8192 octets). Il y aura donc un conflit entre x[i] et y[i] pour une même ligne lorsque l'on voudra calculer x[i] * y[i].

Pour un vecteur de taille N = 2048 :

On a encore le même problème que pour un vecteur de taille N = 512 et 1024 avec un problème en plus pour un cache de taille 16384 octets (2048*8 = 16384).
On peut remarquer que une associativité différente de 1 permet de diminuer ce défaut de conflit dans ces 3 cas de vecteur (N = 512/1024/2048) car un bloc aura 2 ou 4 lignes qui avec la politique de remplacement LRU permet d'éviter le défaut de conflit.

4. Produit matrice-vecteur

Ici, nous calculons le produit d'une matrice et d'un vecteur de taille N.
De même que pour le produit scalaire, on garde le même principe de raisonnement.

On constate comme le produit scalaire que plus la taille de la ligne du cache augmente plus le taux d'erreur diminue.

De même, le problème de la taille du cache qui correspond à la taille du vecteur persiste et l'on observe ainsi un taux d'échec de 1 pour les vecteurs de taille 512 et 1024 pour les configurations avec une associativité de 1, en raison du même cycle des adresses de x et y dans les lignes du cache.

L'augmentation de l'associativité permet de réduire le taux d'erreur. A la différence du produit scalaire on observe un meilleur taux d'échec pour une même configuration, cela est dû au fait que certaines lignes du cache conservent les anciennes données du vecteur y qui demande moins de lignes de cache. Cela réduit ainsi le défaut de cache du vecteur y et donc directement le taux d'échec.

5. Produit de matrices ijk

Ici , nous calculons le produit de matrices carrées Z = X * Y dans l'ordre ijk.

Pour N=16 : Lorsque l'on à 2 lines par set le taux est généralement le plus élevé. Avec une taille de cache plus grande on obtient une stabilité dans le taux d'erreur.

Le taux d'erreur est lié dans cette partie à un soucis dans la boucle interne. Il faut aussi constater que le taux d'échec est lié au nombre d'échecs par itération de la boucle interne.

Il y a aussi un problème de sens de lecture des données, car ici on lit la donnée en colonne alors que la lecture se fait normalement en ligne. d'où les défauts de caches. Ce taux d'échec diminue avec une taille de cache plus grande car on a plus d'espace pour stocker les données.

Expliquons ici le cas où N= 64 : nous avons globalement un taux d'erreur à 0.5 partout. Les taux d'erreur similaires seraient due à la puissance de 2. En effet , les lignes ont des tailles en puissance de 2 c'est pourquoi en remplissant avec des vecteurs en taille de puissance de 2, un certain rapport avec les lignes et les matrices se crée et le cycle cause un écrasement des données un même nombre de fois en cache. Même si N = 16 est aussi une puissance de 2, elle est assez petite pour ne pas suivre le même fonctionnement que N = 64.

6. Produit de matrices ijk après transposition

Nous allons calculer ici le produit de matrices carrées Z = X * Y dans l'ordre ijk après transposition de la matrice Y. Ainsi nous pourrons comparer nos résultats avec la 5 et voir si la transposition aide dans la réduction du taux d'erreur.

Les données sont accédées de manière contiguë, on remarque donc que pour une grande valeur de N , on diminue le taux d'échec.

Pour N=100 on passe en moyenne de 30% à 10% d'erreurs.

Ceci est dû au fait que pour des grands N , le stockage de données voisines évite de devoir recharger à chaque itération.

7. Produit de matrices ikj

Ici nous calculons encore une fois le produit de matrices carrées Z = X * Y mais cette fois ci dans l'ordre ikj.

Nous avons ici , les meilleurs taux d'erreur pour N= 16,64 et 100. C'est le meilleur des cas car nous évitons le problème du produit de matrice ijk en réalisant la boucle k avant la boucle de j. Nous réduisons donc le taux d'erreur pour y[k][j] et ce sans réaliser de transposition.

En mettant en perspective les résultats des parties 5, 6 et 7 nous en déduisons que la meilleure stratégie à adopter pour le calcul d'un produit de matrices serait de choisir l'ordre de produit de matrices ikj.