15 Novembre 2022

Outils de Base pour le HPC : TD 2

Programmation C et mesures de performances

ROUSSELLE Pacôme

Métrique mesurée : Bande Passante (MiB/s). On cherche à voir quelle quantité de données sont traitées en secondes à l'exécution.

Condition de stabilité des mesures : Un écart-type inférieur à 5%

Taille Matrice/Vecteurs : n = 100.

Au début du travail, je voulais faire des mesures sur des tailles de données croissantes, mais j'ai préféré rester sur une même taille pour tous les kernels et que je considère assez importante pour apercevoir des différences notables entre version et compilateur.

Taille du problème pour chaque kernel :

On manipule des flottants en double précision => 64 bits => 8 octets par flottant.

Registres : 64B, 0 à 1 cycle pour y accéder => 1 flottant max

Cache L1 = 32KB, environ 4 cycles pour y accéder => 4000 flottants max

Cache L2 = 256KB ~10 cycles d'accès => 4000 < n < 32000 flottants

D'où pour n = 100 :

Dgemm : Produit de matrices de flottants A[nn] x B[nn] = C[nn]

100x100x8x3 = 240 KB => cache L2

Dotprod : Produit de vecteurs de flottants a[n] x b[n] = c[n]

100x8x3 = 2,4KB => cache L1

Reduc : Réduction d'un vecteur de flottants a[n] à un flottant d de taille 8B

100x8 + 8 = 808B => cache L1

Cette différence de stockage en cache explique les différences de bande passante que l'on observe entre dgemm et dotprod/reduc.

Il faut noter que tous les vecteurs et matrices sont déclarés « inline », ce qui signifie que leurs accès mémoires sont reliés les uns à la suite des autres. Il n'y a donc pas de temps supplémentaires pour chercher leurs données dans la mémoire.

Dgemm : c[ij] = a[ik]*b[kj]

Version IJK :

Accès Matrice A : Ligne

Accès Matrice B : Colonne, ce qui fait avancer la lecture des données de B de n octets à chaque tour de boucle, impactant la vitesse de calcul.

Accès Matrice C : Invariant

Version IKJ :

Accès Matrice A : Invariant

Accès Matrice B : Ligne

Accès Matrice C : Ligne

IKJ ne fait aucun accès colonne, ce qui permet de lire n+1 octets en ligne, sans saut de n pour écrire dans n octets, ce qui est mieux que IJK

Version IEX :

Accès Matrice A : Invariant => Extrait, plus besoin d'accéder à la case [i*n+k]

Accès Matrice B : Ligne

Accès Matrice C : Ligne

Avec l'extraction on réduit le temps de IKJ.

Version Unroll 4 et 8 : Aucune amélioration notable à partir du flag -O2, on fait toujours le même nombre d'accès mémoire qu'en IEX, juste moins d'appels de boucle. Ce nombre réduit d'appels peut expliquer les légères améliorations pour une compilation sans optimisation ainsi que pour le flag -O1.

Version Cblas : Fonction optimisée pour le calcul linéaire, donnant la meilleure bande passante.

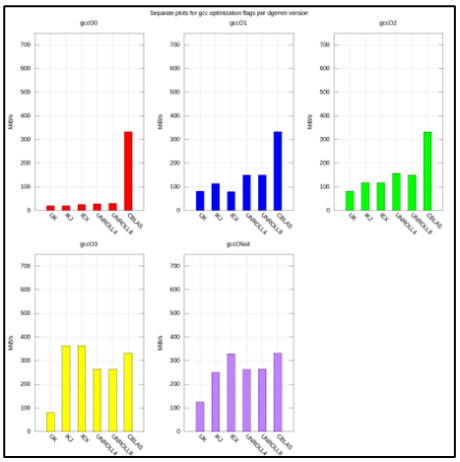

Fig 1 : Histogrammes pour compilation avec gcc

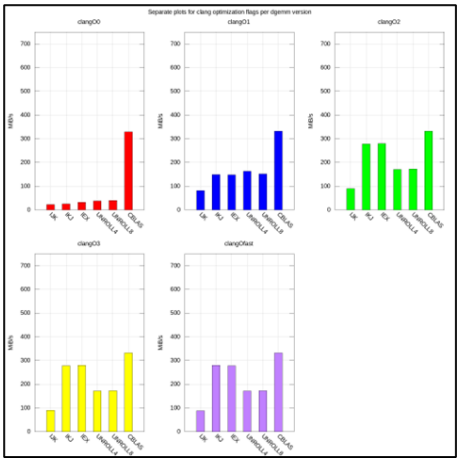

Fig 2 : Histogrammes pour compilation avec clang

Dotprod : c += a[i]*b[i]

La taille de données du problème pour le même n comparé à dgemm est plus petite d'un facteur 100, ce qui permet de le stocker dans le cache L1, réduisant ainsi le nombre de cycles processeuers pour accéder aux données.

Base :

Contrairement au kernel dgemm, les données sont organisées en ligne. On ne fait que lire 2 vecteurs de n valeurs et écrire n fois dans un troisième vecteur.

Unroll :

Comme dans dgemm, il n'y a aucune amélioration notable à partir du flag -Ofast. On fait le même nombre d'accès mémoire, juste moins de tours de boucle. Le nombre réduit d'appels peut expliquer les légères améliorations pour les compilations avec les flags précédant -Ofast.

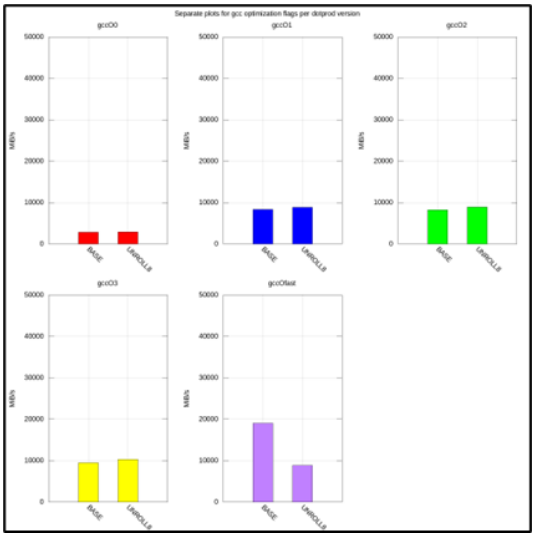

Fig 3 : Histogrammes pour compilation avec gcc

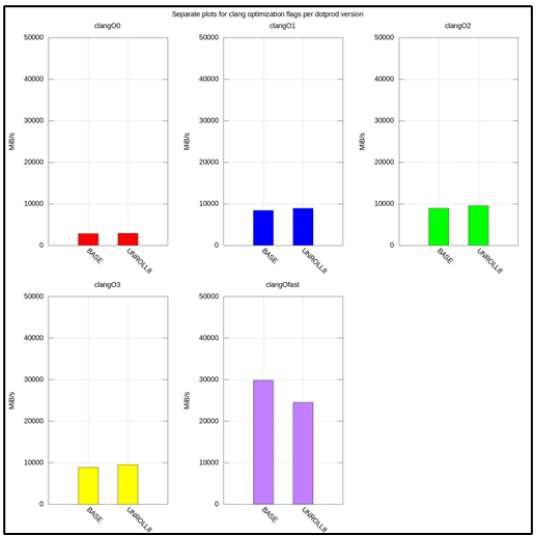

Fig 4 : Histogrammes pour compilation avec clang

Reduc : d += a[i]

La taille de données du problème pour le même n comparé à dgemm est plus petite d'un facteur 300 (facteur 3 par rapport à dotprod), ce qui permet de le stocker dans le cache L1, réduisant ainsi le nombre de cycles processeurs pour accéder aux données.

Base :

Tout comme avec le kernel dotprod, les données sont organisées en ligne. La performance est cependant légèrement supérieure car on ne lit qu'un seul vecteur au lieu de deux (n lectures au lieu de 2n).

Unroll :

Comme dans dgemm, il n'y a aucune amélioration notable à partir du flag -O3. On fait le même nombre d'accès mémoire, juste moins de tours de boucle. Le nombre réduit d'appels peut expliquer les légères améliorations pour les compilations avec les flags précédant -O3.

On notera la nette amélioration avec clang -Ofast.

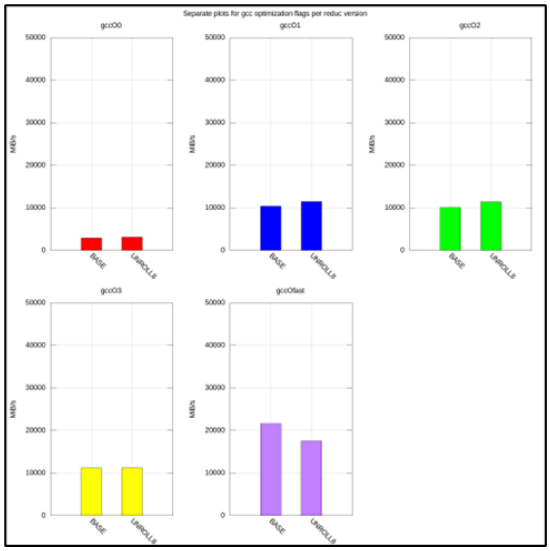

Fig 5 : Histogrammes pour compilation avec gcc

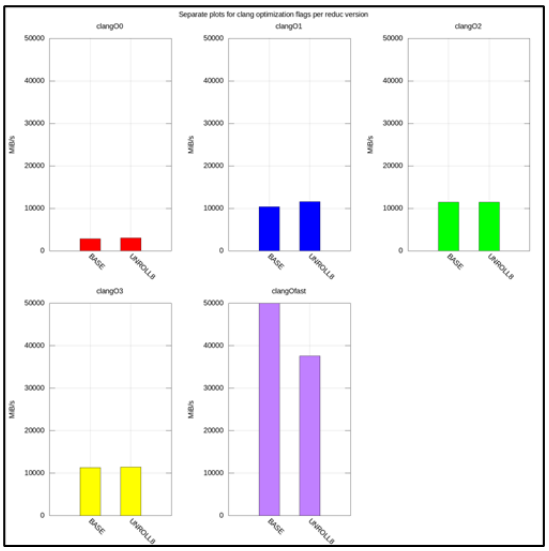

Fig 6 : Histogrammes pour compilation avec clang