Rapport OBHPC DGEMM

SANCHEZ-GUENRO Maëlo

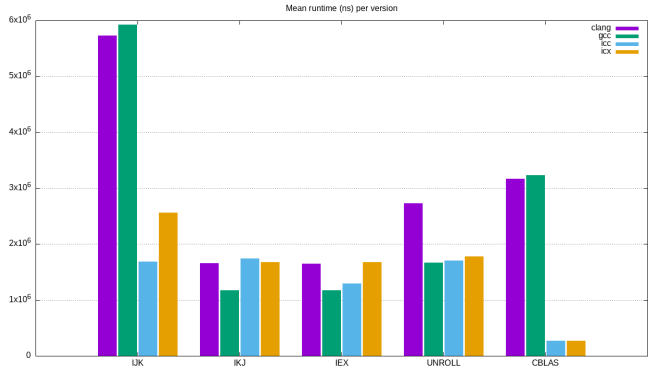
Novembre 2022

1 Environnement et exécution

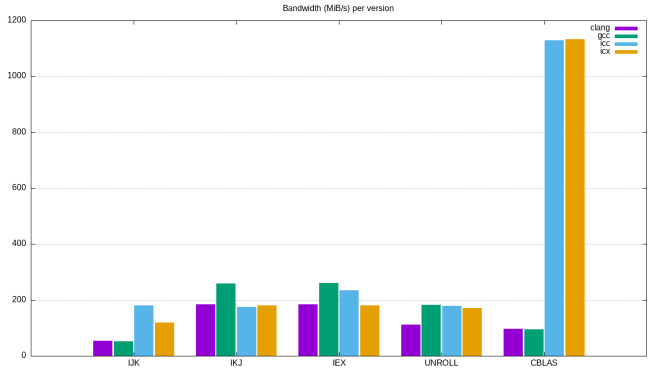
Toutes les mesures ont été réalisée sur le même ordinateur, branché au secteur, sur le même coeur sous gouverneur performance, et avec les mêmes versions de compilateurs.
Les mesures effectuées pour comparer la vitesse des différentes implémentations de la dgemm ont été menées sur des matrices 200 par 200 avec 20 itérations. J'ai utilisé cette taille car elle permet un temps d'exécution suffisamment long pour niveler les instabilités d'une exécution tout en rentrant dans le cache l2 ce qui permet que le temps d'exécution de soit pas trop long non plus et faire plus de répétitions

2 Comparer les version initiales

Ici j'ai utilisé seulement le code de base, compilé avec les 4 compilateurs icc, icx, gcc, clang et le flag d'optimisation -O3. Le premier constat est que le compilateur intel et très efficace par rapport à clang et gcc.
Dans le cas de la version naïves du dgemm, en ijk, le temps d'exécution des versions icc et icx est de l'ordre de 50% des deux autres compilateurs ce qui est d'un ordre similaire aux autres implémentations manuelles. D'ailleurs, en regardant aussi la bande passante, on remarque que l'ancien compilateur intel est au moins aussi voir plus performant que le nouveau dans chaque cas .
Dans le cas de l'implémentation utilisant la fonction de la bibliothèque blas, on voit un énorme écart, d'un coté cette implémentation n'est que la deuxième moins performante alors que du coté d'intel c'est de loin la meilleur. C'est la seul qui descend largement en dessous de la ms, quelque chose de l'ordre de 100-200µs,tandis que la bande passante dépasse le GiB/s qu'aucune autre ne vas au-dessus des 300 MiB/s
Au vu des ces résultats, on peut en déduire que les compilateurs intel appliquent probablement directement une des méthodes qui a été implémenté manuellement lorsqu'il rencontre une version naïve de l'algo. Aussi, la différence avec blas montre qu'ils profitent fortement que celle-ce soit directement implémenté et optimisée dans me mkl de intel.



(a) Temps d'exécution moyen en ns pas version



(b) Bande passante en MiB/s par version

Il faut tout de même remarqué que l'écart type de cblas avec intel est très important, de l'ordre de 50%. Cela indique que selon les données à traiter le temps d'éxécutiosn de vette version peut beaucoup varié même s'il reste le plus bas

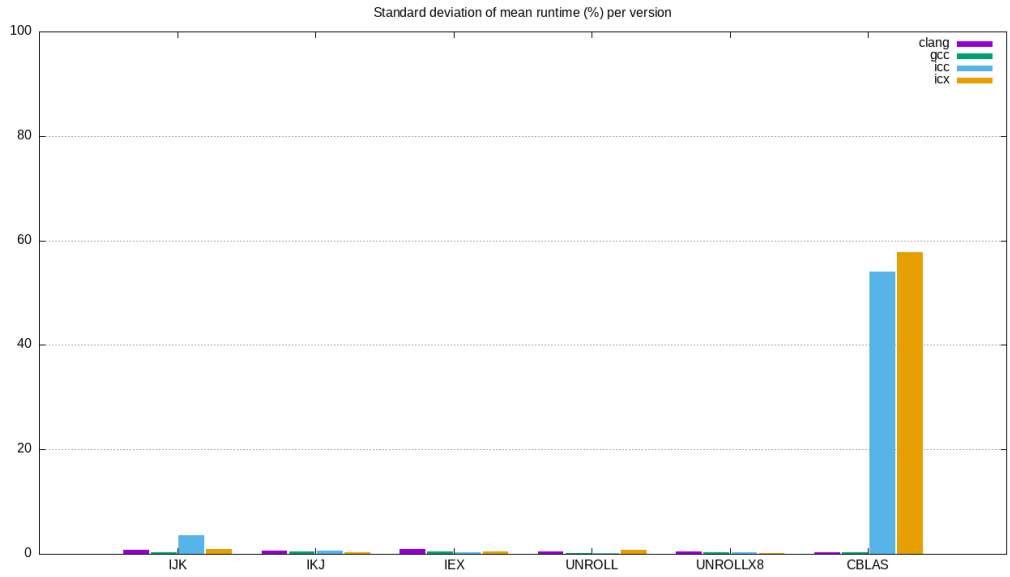
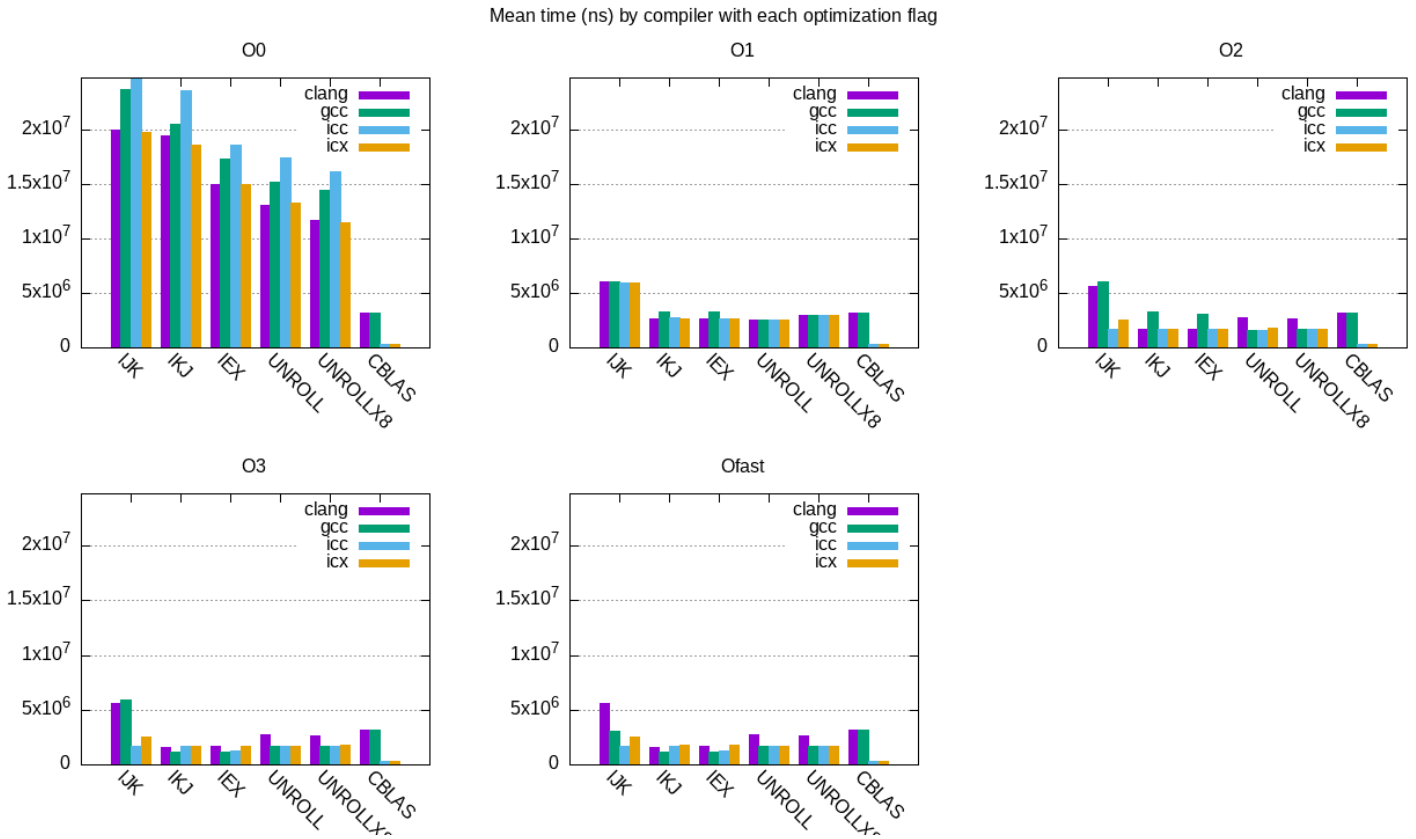


Figure 1: Écart type avec le temps d'exécution moyen en pourcentage par implémentation (-O3)

Sinon, en regardant dans l'ensemble ce graphe excepté blas, il semble que l'implémentation la plus efficace soit IEX, ce qui veut dire que l'ajout d'un déroulement de boucle manuel a fait plus de mal que de bien à l'implémentation le contenant. Et cela se voit pour tout les flags d'optimisation sauf 1.



Pourtant, on voit bien qu'avec le flag -O0, chaque version est meilleur que la précédente. Ce flag est en fait une référence pour savoir si une méthode utilisé est meilleur que les autres individuellement car il ne fournis aucune optimisation d compilateur. Donc le déroulage de boucle est une bonne méthode pour améliorer les performances, mais n'est pas la meilleur implémentation avec toutes les optimisations, car que le compilateur fait un meilleur travail à déroule des boucles de lui-même.Les meilleurs conditions, dans les implémentations données, pour qu'il fasse ce travail sont donc présentes avec la version IEX.

3 Et si on déroule plus ?

Quand on regarde la figure juste ci-dessus, passer d'un déroulage de 4 à 8 semble ne pas affecté significativement le programme, à part sans optimisation processeur. On remarque tout de même et on le constate en se concentrant sur ces deux implémentations que clang est très mauvais pour gérer des fonctions déroulés manuellement. Si on regarde l'évolution du temps d'exécution cependant, celui-ci à diminué chez clang alors qu'il est resté constant voir à augmenté pour les autres. Les compilateurs sont sûrement "habitués" à traité des déroulement de 4, ou gère mieux un déroulement déjà présent s'il est plus faible.

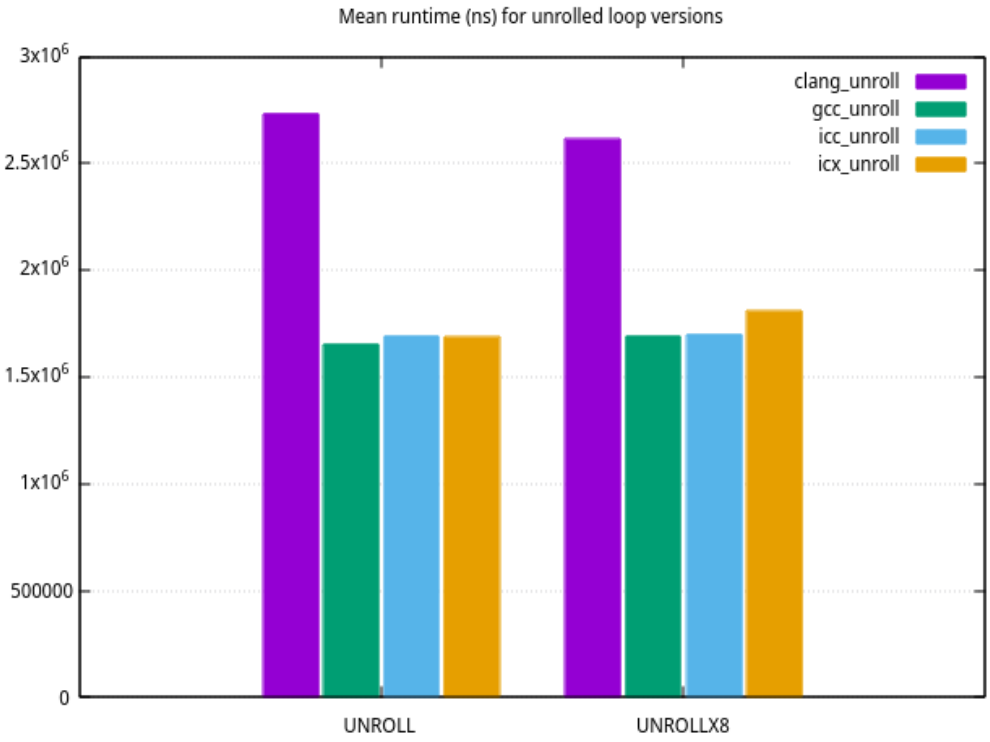


Figure 2: Temps d'exécution moyen des déroulages ed boucles (ns)

4 Dotprod

Pour le dotprod, j'ai essayé de dérouler la boucle, mais cela change l'odre des additions et donc potentiellement le résultat final de la fonction. Les résultats montre que (insérer résultats obtenus)