ROUSSELLE Pacôme

N°21902985

Outils de Base pour le HPC : TD 2

Programmation C et mesures de performances

Table des matières

Introduction :

Cette introduction expose la méthode employée pour répondre aux questions du TD, avant de décrire la structure des parties consacrées aux codes dgemm, dotprod et reduc.

Données analysées :

On cherche ici à comparer la vitesse de chaque version, c'est-à-dire le nombre de MiB/s mesurés.

Automatisation :

Afin de pouvoir répéter les mesures et rendre le procédé portable, la production de fichiers de performance et de leurs graphiques associés est automatisée au moyen de scripts bash, deux par sous-dossier, avec un script central dans le dossier parent :

Le script « run » fixe la taille des matrices à multiplier et crée les fichiers de performance de chaque compilateur par versions (les flags d'optimisation ont leur propre fichier). Tous les lancements d'exécutables sont successivement affectés au cœur 1 via la commande taskset.

Le script « data » crée les fichiers de performance pour chaque version avec grep et les place dans un sous-dossier dédié. Les fichiers .dat des compilateurs sont déplacés dans un autre sous-dossier dédié. Il utilise ensuite les scripts gnuplot présents dans le sous-dossier plot pour produire les graphiques correspondants à chaque fichier de performance, avant de les stocker dans le sous-dossier graphs.

Le script central perf.sh rentre successivement dans chaque sous-dossier avant d'appeler leurs scripts run et data, ce après avoir fixé la fréquence du cœur 1 sur le gouverneur performance avec cpupower. La fréquence est fixée à nouveau avant chaque entrée dans un sous-dossier.

Makefile :

Les seuls ajouts au Makefile de chaque code consistent en l'ajout de quatre lignes de compilation, une pour chaque flag d'optimisation, suivi de la création de l'exécutable associé. Ainsi, compiler le code source de dgemm avec un flag O2 produit l'exécutable dgemmO2.

Informations sur l'architecture cible :

Les informations du hardware sur lequel les mesures ont été effectuées, prises à l'aide des commandes lscpu et cat sont stockées dans les fichiers texte cpuinfo, lscpu (question 0.1) et registers (question 0.2).

Dgemm

Gcc :

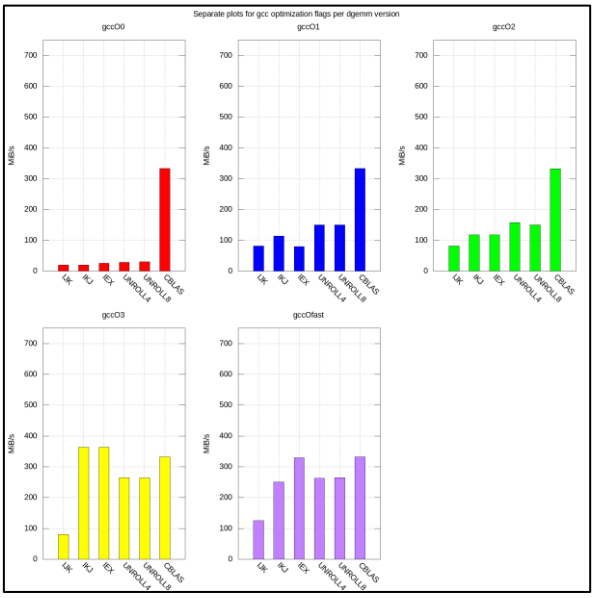

Fig. 1 – Comparaison des performances pour le compilateur gcc sur le code dgemm, avec cinq flags différents d'optimisation

a) Observations générales :

- La vitesse du programme évolue de manière inversement proportionnelle par rapport à la complexité de l'algorithme de calcul, ce qui est mathématiquement logique en raison du nombre de calculs réduit.

- La vitesse du programme augmente également à mesure que les flags d'optimisation sont plus performants.

- La version cblas est largement plus rapide sur les flags -O0 à -O2, sûrement en raison des optimisations internes de la fonction. Sa vitesse est également constante car il s'agit d'un appel à la même fonction.

- Les versions à déroulage à 4 et 8 ont des performances quasiment égales entre elles, mais ne présente une amélioration concrète des performances que sur les flags -O0 à -O2.

b) Interrogations sur les résultats :

- Pourquoi les versions à déroulage perdent-elles en intérêt sur les flags -O3 et -Ofast ? Je pense que cela est dû à la gestion des miettes, faisant rentrer le programme dans une seconde boucle et faisant perdre du temps par rapport aux autres versions.

- Le flag -03 donne les mêmes performances pour IJK et IEX, tandis que -Ofast fait baisser la vitesse de IJK, en gardant IEX à une vitesse similaire. Il s'agit peut-être de spécificités de ses flags, expliquant également la légère baisse d'IEX en -01. Une autre explication peut venir d'un problème de mesures.

Clang :

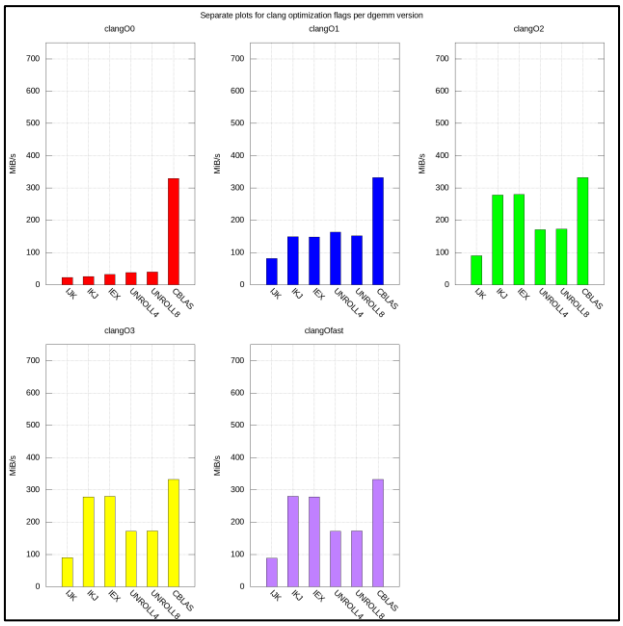

Fig. 2 – Comparaison des performances pour le compilateur clang sur le code dgemm, avec cinq flags différents d'optimisation

a) Observations générales :

- Les résultats étant très similaires à gcc, les cas particuliers sont traités dans la section suivante.

b) Interrogations sur les résultats :

- Les versions IJK et IEX deviennent nettement plus rapides à partir du flag -O2, et restent ensuite constantes. Est-ce que ces flags ont des composantes gérant plus efficacement les boucles ?

- Les performances de -O2 et -Ofast sont identiques. Je pense avoir fait une erreur de mesures.

- IEX en flag -O1 a une performance similaire aux autres versions. Je pense avoir fait une erreur de mesures.

Dotprod

Gcc :

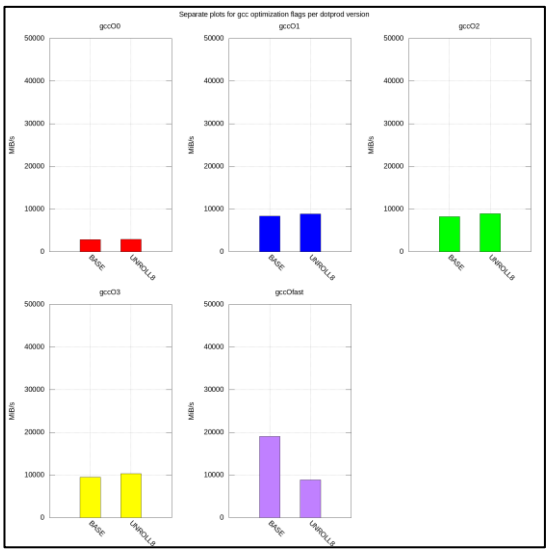

Fig. 3 – Comparaison des performances pour le compilateur gcc sur le code dotprod, avec cinq flags différents d'optimisation

a) Observations générales :

- Le calcul dotprod étant moins complexe que dgemm, on obtient naturellement une performance accélérée.

- La version à déroulage x8 n'apporte pas d'amélioration significative pour les flags -O0 à -O3, et est même complétement inutile avec -Ofast.

- Les performances n'évolue pas de -01 à -03.

b) Interrogations sur les résultats :

- De même que pour le code dgemm, je pense que la gestion des miettes allonge fortement le temps d'exécution du déroulage.

Clang :

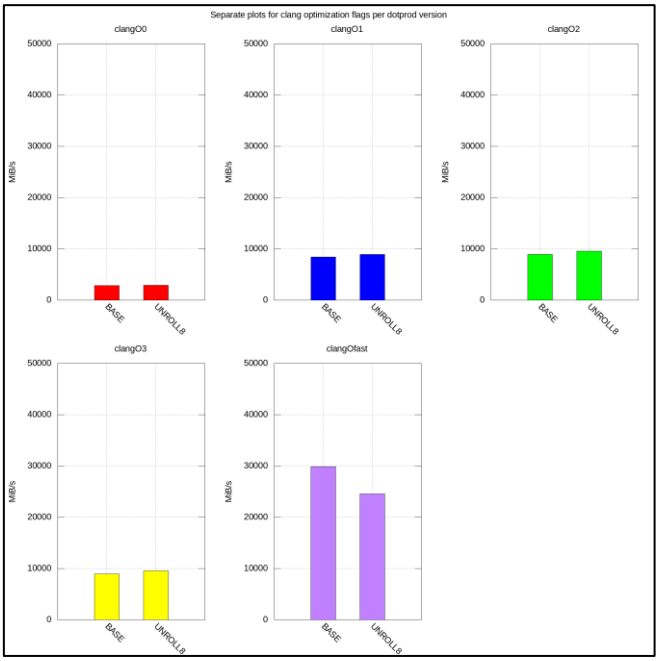

Fig. 4 – Comparaison des performances pour le compilateur clang sur le code dotprod, avec cinq flags différents d'optimisation

a) Observations générales :

Les résultats sont très similaires avec gcc : Une performance globale accélérée due à l'amélioration de la complexité, la version à déroulage x8 n'apportant pas d'amélioration significative pour les flags -O0 à -O3, et reste complétement inutile avec -Ofast. Les performances n'évoluent pas non plus de -01 à -03.

b) Interrogations sur les résultats :

- Cependant je ne comprends pas comment la performance avec -Ofast augmente de 1000 MiB/s par rapport à gcc. Peut-être une intégration spécifique pour les produits.

- De même que pour le code dgemm et la compilation de dotprod par gcc, je pense que la gestion des miettes allonge fortement le temps d'exécution du déroulage.

Reduc

Gcc :

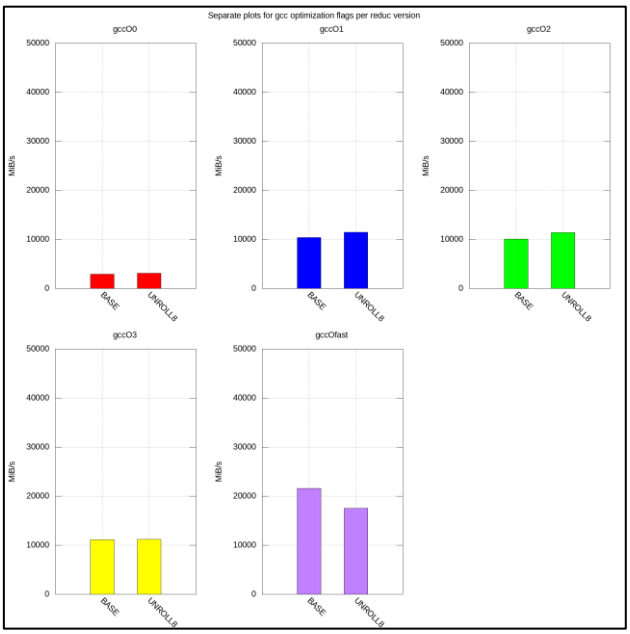

Fig. 5 – Comparaison des performances pour le compilateur gcc sur le code reduc, avec cinq flags différents d'optimisation

a) Observations générales :

Il y a peu à dire comparé à dotprod : la complexité est légèrement meilleure, augmentant donc de peu la performance. Les flags -O1 à -O3 ont des résultats similaires, voire égaux.

b) Interrogations sur les résultats :

Cependant, la version à déroulage x8 a doublé sa performance avec -Ofast, réduisant son écart avec la version de base. Je pense que cela est simplement dû à l'amélioration de la complexité, plutôt qu'à une amélioration de l'implémentation de la fonction.

Clang :

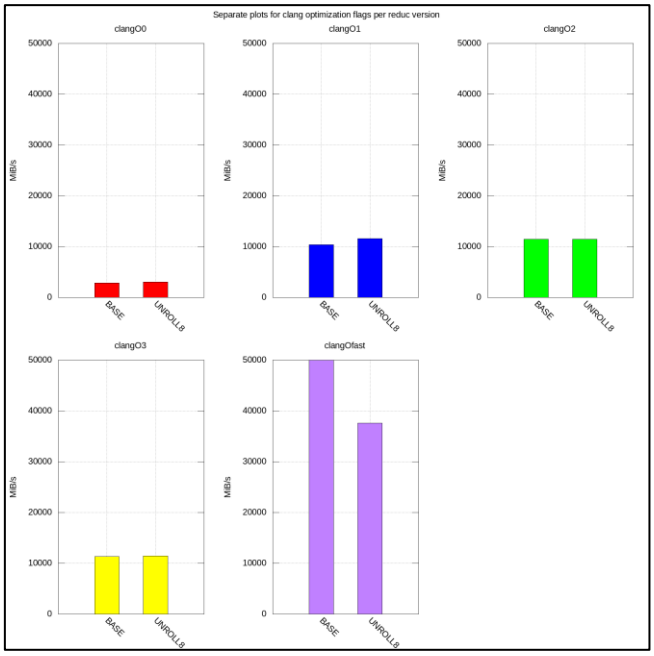

Fig. 6 – Comparaison des performances pour le compilateur clang sur le code reduc, avec cinq flags différents d'optimisation

a) Observations générales :

Comme pour gcc, il y a peu à dire comparé à dotprod : la complexité est légèrement meilleure, augmentant donc de peu la performance. Les flags -O1 à -O3 ont des résultats similaires, voire égaux.

b) Interrogations sur les résultats :

Pourquoi les deux versions avec -Ofast voient-elles ses performances augmenter autant (l'histogramme sort du graphique mais la dernière mesure donne 52072.393 MiB/s) ? L'écart-type à 5.234 % (seul écart-type supérieur à 5% de toutes les mesures, à chaque batterie de mesures) me conforte dans l'idée que c'est une erreur de mesure.

Améliorations possibles

Les améliorations du code et des mesures proposées ici ont été envisagées, mais non implémentées par manque de temps, en majorité dû aux difficultés rencontrées à l'automatisation des mesures et l'apprentissage des scripts gnuplot.

- Augmentation du nombre de compilateurs : J'aurais voulu pouvoir mesurer les performances des compilateurs icc et icx (les commandes pour produire les fichiers de performance sont en commentaires dans chaque script run). Ceci n'est plus possible car je n'aurai pas accès à un système Linux avec icc et icx installés à temps pour la date de rendu.

- Augmentation du nombre de versions : Des versions à déroulage x4, ainsi qu'une version CBLAS étaient prévues pour les codes dotprod et reduc, afin d'avoir un plus grand nombre d'histogrammes et de comparaisons. Ceci n'est plus possible car je n'aurai pas accès à un système Linux à temps pour la date de rendu.