UVSQ
université PARIS-SACLAY

Rapport :
TD2 OBHPC

Myriam
Hamidou

M1 CHPS

UVSQ
université PARIS-SACLAY

Introduction

Grâce à ce TD on analyse la performance, des fonctions dgemm, dotprod et reduc implémentées à la main en les comparant à les fonctions de la bibliothèque de CBLAS.

Pour cela plusieurs versions des fonctions ont étaient implémentées, des versions qui sont de plus en plus optimisées. Le but est d'obtenir une version le plus optimisée possible des fonctions. Cela passe aussi par le choix du flags d'optimisation lors de la compilation mais aussi par le choix du compilateur.

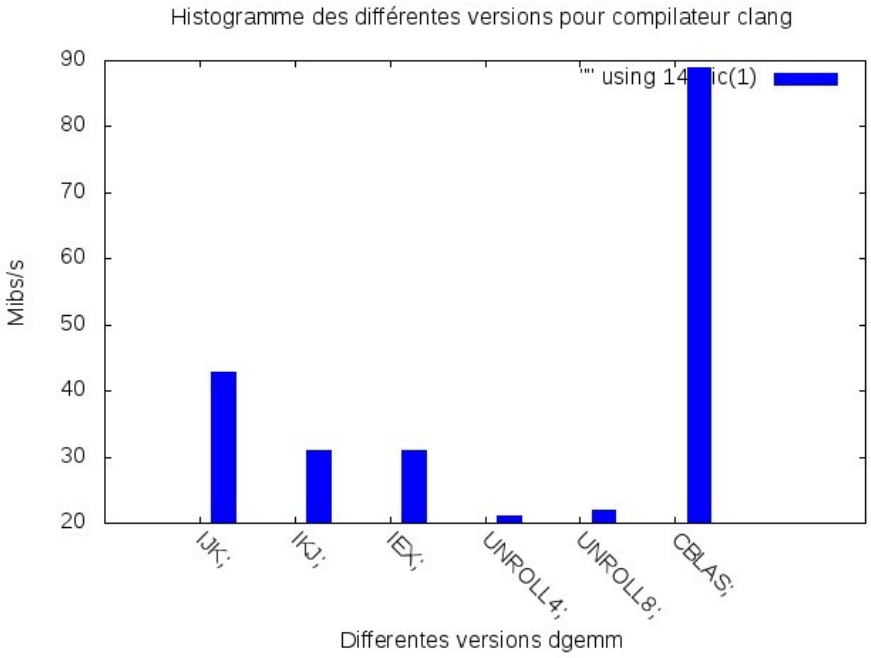
Pour analyser ces codes et obtenir la meilleur performance, le laptop est connecté au secteur et on s'assure que le CPU tourne à des fréquences stable.

Pour en savoir plus sur le CPU du laptop on récupère ces information :

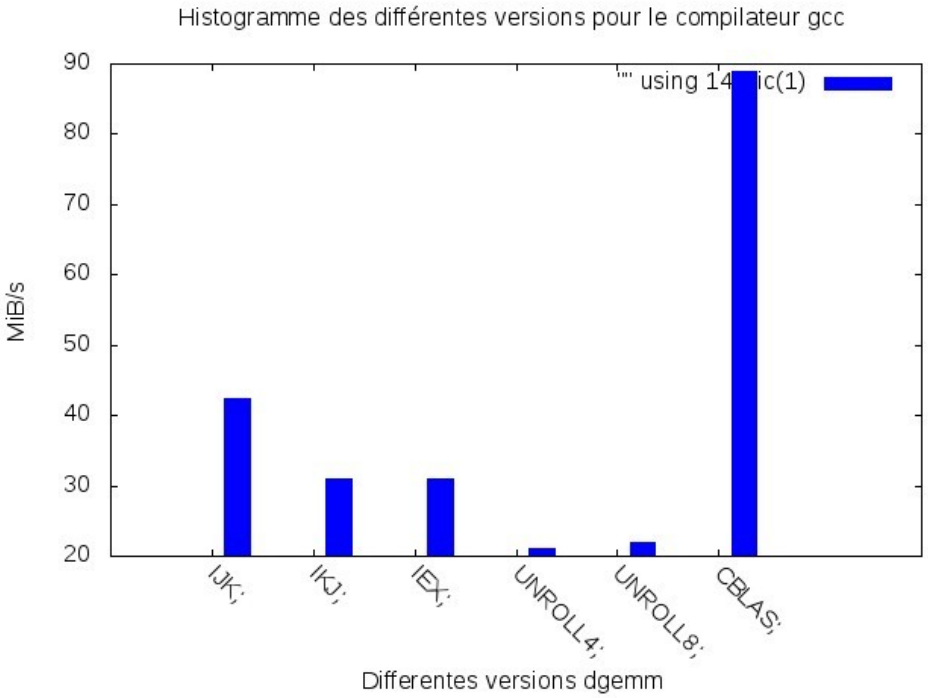
-Architecture : x86_64
-Mode(s) opératoire(s) des processeurs : 32-bit, 64-bit
-Boutisme : Little Endian-
-Processeur(s) : 2
-Liste de processeur(s) en ligne : 0,1
-Thread(s) par cœur : 1
-Cœur(s) par socket : 2
-Socket(s) : 1
-Nœud(s) NUMA : 1
-Identifiant constructeur : GenuineIntel
-Famille de processeur : 6
-Modèle : 55
-Nom de modèle : Intel(R) Celeron(R) CPU N2840 @ 2.16GHz
-Révision : 8
-Vitesse du processeur en MHz : 2342.532
-Vitesse maximale du processeur en MHz : 2582,3000
-Vitesse minimale du processeur en MHz : 499,8000
-BogoMIPS : 4326.40
-Virtualisation : VT-x
-Cache L1d : 24K
-Cache L1i : 32K
-Cache L2 : 1024K
-Nœud NUMA 0 de processeur(s) : 0,v1

Fonction : dgemm

Pour commencer comparons les versions de dgemm pour chaque compilateur. Pour cela on pose n=100 et r=80. On obtiens alors l'histogramme suivant pour le compilateur clang :

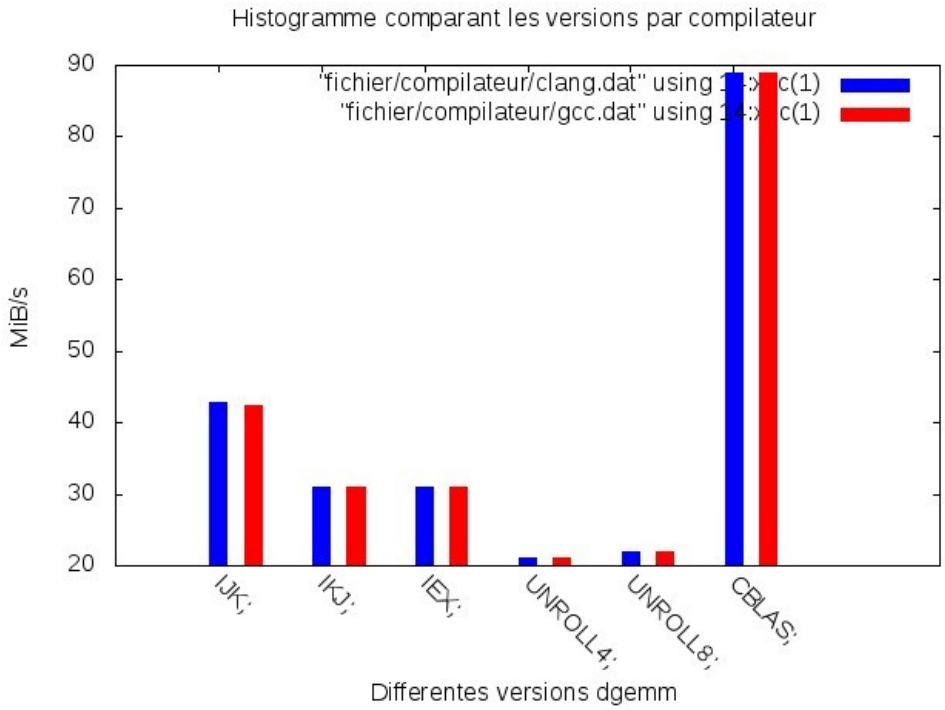


Et celui-ci, pour le compilateur gcc :



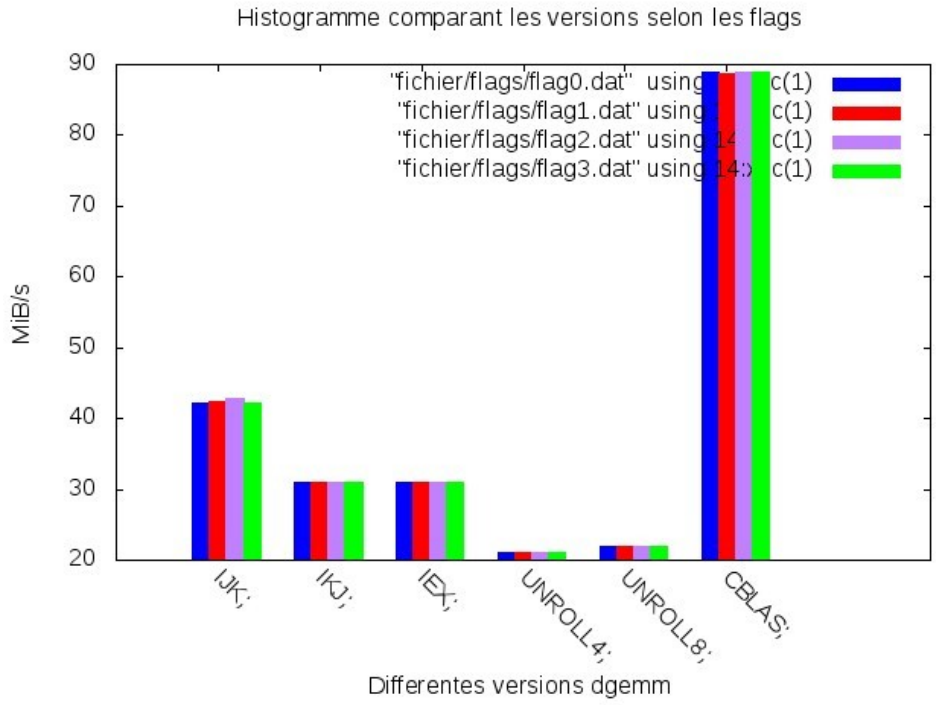
On devrait normalement observer que plus le code est optimisé et plus ces performances sont meilleures, ce qui n'est pas le cas ici, peut-être du a une frequence du CPU pas assez stable. Mais comme prévu, CBLAS a une performance beaucoup plus élevé que les codes implémentée. On remarque aussi que le code deroulage x8 à une meilleure optimisation que le déroulage x4 car plus on augmente le déroulage et plus on augmente le parallélisme des instructions et donc augmente les performances.

Maintenant comparons les versions selon les compilateur gcc et clang. On obtient l'histogramme ci-dessous :



Pour la version non optimisé et UNROLL , clang a est légèrement plus performante que gcc mais pour les autres versions en particulier CBLAS, gcc est plus performantes que clang.

Ensuite comparons les versions selon les flags d'optimisation. On obtiens l'histogramme suivant :

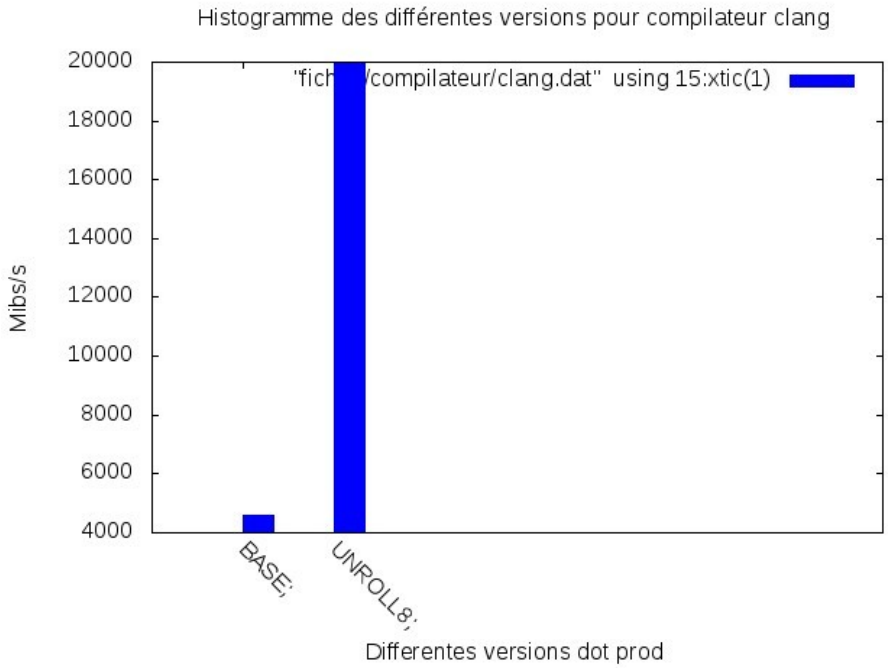


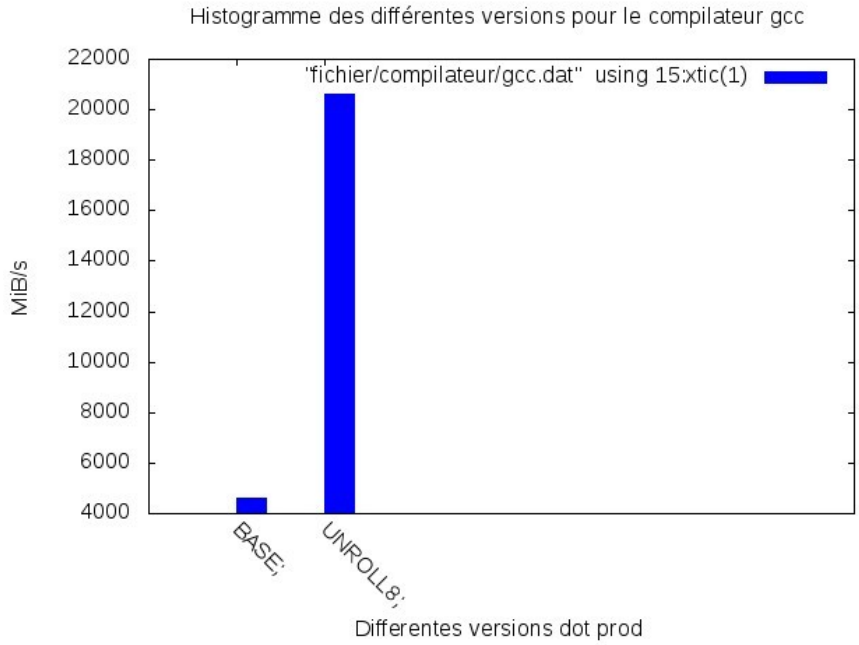
Selon la version du code les flags les plus performante ne sont pas les même. En effet les différentes flags n'optimise pas de la même façon. Par exemple le flags d'optimisation -O0 n'optimise pas du tout le code, le flags -O1 est le premier niveau d'optimisation, et donc essaye d'accéléré le code tout en le rendant plus compact et en réduisant le temps de compilation. Le flags -O2 en plus de faire les même chose que le flags -O1, elle essaye d'augmenter la performance sans compromettre la taille du code, aura donc un temps de compilation un peu plus long que celui du flags -O1. Et enfin le flags -O3 augmente la performance selon le code mais augment le temps de compilation et la taille en mémoire du code. Ils faut donc adapter les flags selon les versions pour obtenir une meilleur optimisation.

Fonction:Dotprod et Reduc

On execute les codes avec une matrice de taille 1024 et une répétition de 1000

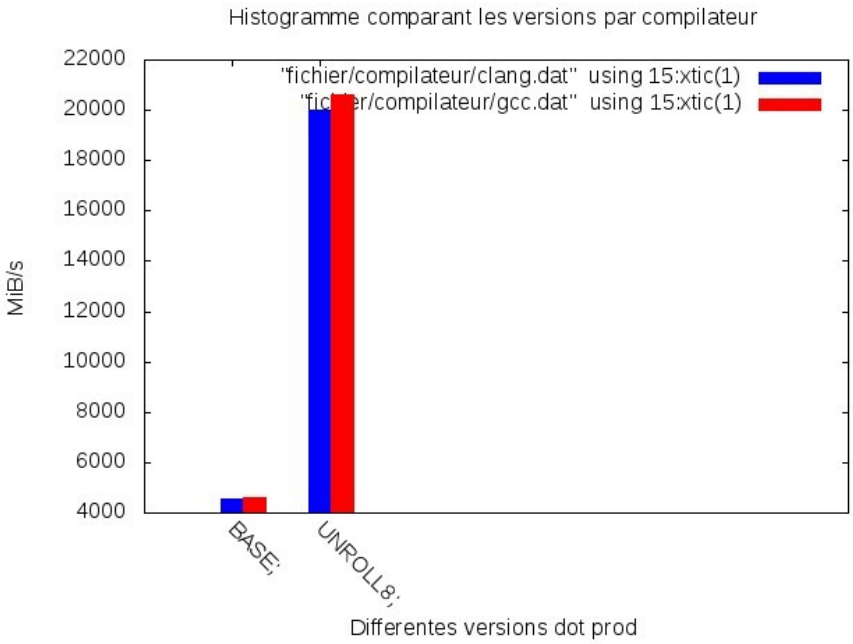
Pour la fonction dotprod et reduc nous comparons la versions basique non optimisée à une version déroulage x8 et obtenons résultat similaire pour dotprod et reduc . Ce qui nous donne les histogrammes suivants pour chaque compilateur :





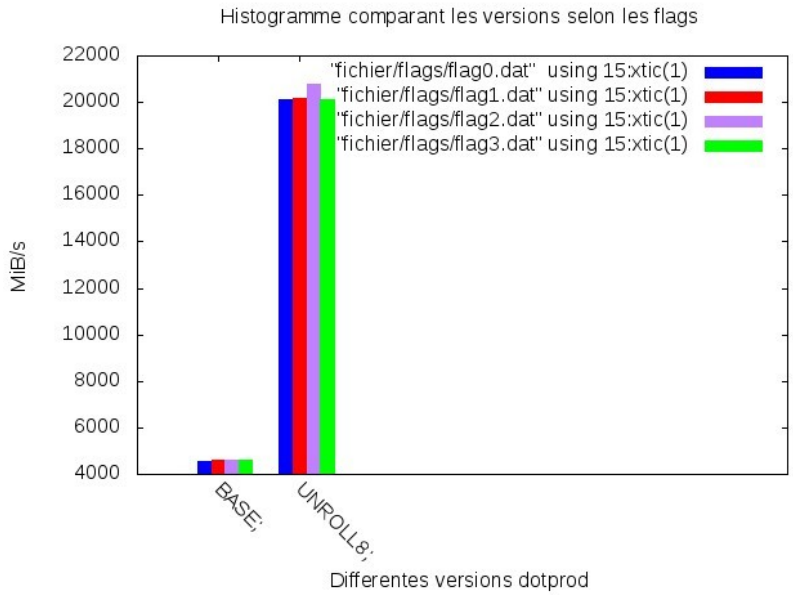
Observe bien une grande amélioration de performance pour la versions de déroulage x8 quelle que soit le compilateur.

Maintenant pour l'histogramme des versions selon les compilateur on obtient ceux-ci :



Pour les deux versions gcc à une meilleure performance que clang.

Pour les flags on obtient l'histogramme suivant :



Comme pour la fonction dgemm, la meilleure performance selon le flags dépend de la version, ici pour la fonction UNROLL 8, le flags -O2 permet d'obtenir des meilleure résultat.