

Intitulé du projet :

Programmation C et mesures de performances OBHPC

Réalisé par:

CHIKHI Katia

Année académique 2022-2023

Table de matières

1	Les contraintes nécessaires pour une mesure de performance stable	3
2	Information sur l'architecture cible	3
3	Description des résultats	4

Liste des figures

1	Figure qui illustre les informations sur le cpu.	3
2	Figure qui illustre les informations sur les caches de données.	4
3	Figure qui illustre les différentes versions de dgemv par compilateur.	5
4	Figure qui illustre la différence entre les compilateurs pour dotprod.	5
5	Figure qui illustre la différence entre les compilateurs pour reduc.	6
6	Figure qui illustre la différence entre les différents flags d'optimisation pour clang.	7
7	Figure qui illustre la différence entre les différents flags d'optimisation pour gcc.	8

1 Les contraintes nécessaires pour une mesure de performance stable

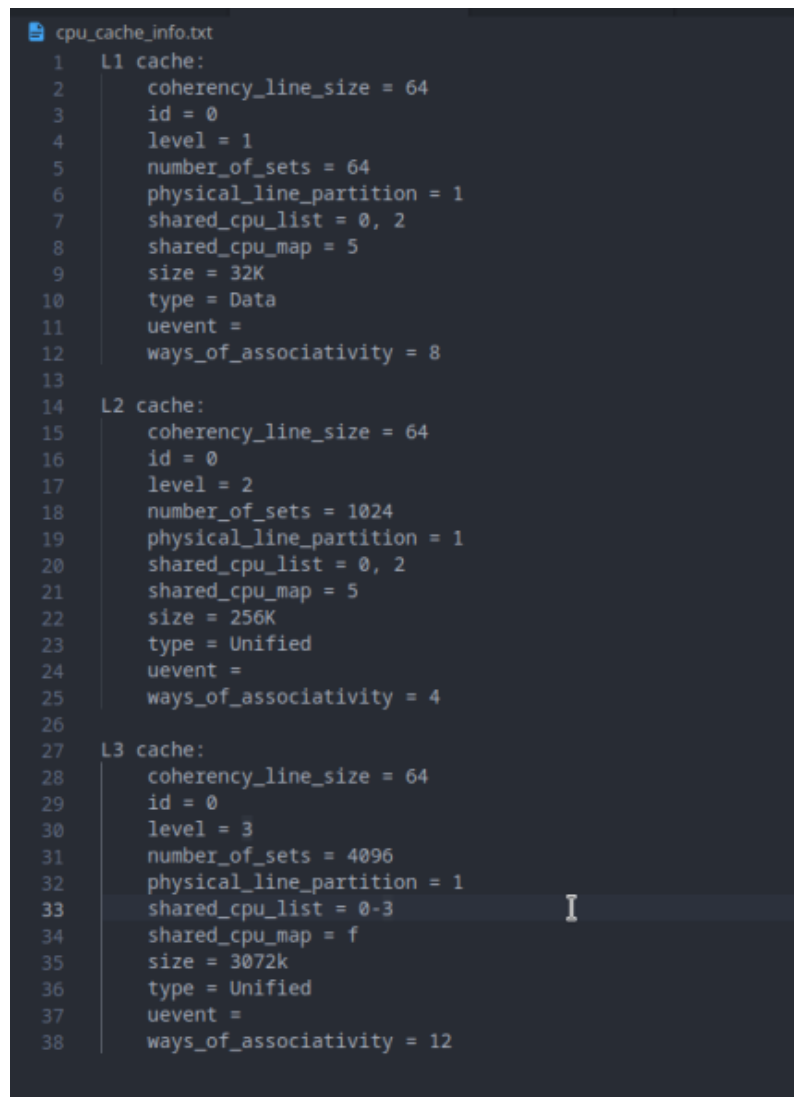
- S'assurer que le laptop est connecté au serveur.
- S'assurer que le CPU tourne a une fréquence stable:
En utilisant cpupower frequency-info, la fréquence du CPU doit être entre 400 MHz et 3,10 GHz, puisque la fréquence actuelle est de 612 MHz alors le CPU tourne a une fréquence stable.
- Pinner le processus sur un cœur de calcul: Les processus sont dans le cœur numéro 0 du CPU.

2 Information sur l'architecture cible

- **Informations sur le CPU** . lscpu rassemble les informations sur l'architecture du processeur à partir de **sysfs** , **/proc/cpuinfo** et de toutes les bibliothèques applicables spécifiques à l'architecture
la figure ci-dessous montre ces informations.

Figure 1: Figure qui illustre les informations sur le cpu.

- **Informations sur les caches de données** La figure ci-dessous montre les informations sur les caches de données.



```
cpu_cache_info.txt
1  L1 cache:
2    coherency_line_size = 64
3    id = 0
4    level = 1
5    number_of_sets = 64
6    physical_line_partition = 1
7    shared_cpu_list = 0, 2
8    shared_cpu_map = 5
9    size = 32K
10   type = Data
11   uevent =
12   ways_of_associativity = 8
13
14  L2 cache:
15    coherency_line_size = 64
16    id = 0
17    level = 2
18    number_of_sets = 1024
19    physical_line_partition = 1
20    shared_cpu_list = 0, 2
21    shared_cpu_map = 5
22    size = 256K
23    type = Unified
24    uevent =
25    ways_of_associativity = 4
26
27  L3 cache:
28    coherency_line_size = 64
29    id = 0
30    level = 3
31    number_of_sets = 4096
32    physical_line_partition = 1
33    shared_cpu_list = 0-3
34    shared_cpu_map = f
35    size = 3072k
36    type = Unified
37    uevent =
38    ways_of_associativity = 12
```

Figure 2: Figure qui illustre les informations sur les caches de données.

3 Description des résultats

La figure ci-dessous montre que :

- Dans l'ensemble, cblas dgemm a donné des performances beaucoup plus élevée que les autres versions. Elle a probablement appliqué l'optimisation la plus avancée pour améliorer les performances

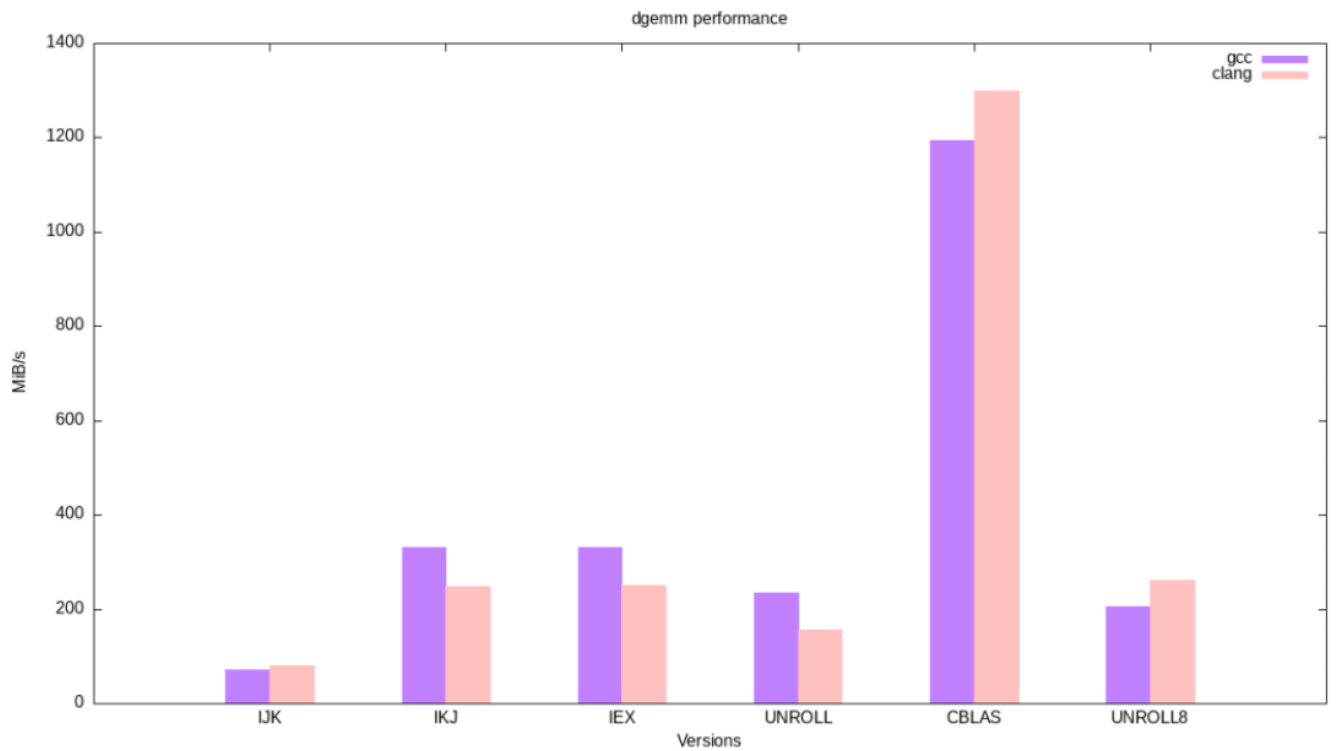


Figure 3: Figure qui illustre les différentes versions de dgemm par compilateur.

La figure ci-dessous montre que le compilateur gcc a donné de meilleurs résultats de performance que clang pour dotprod.

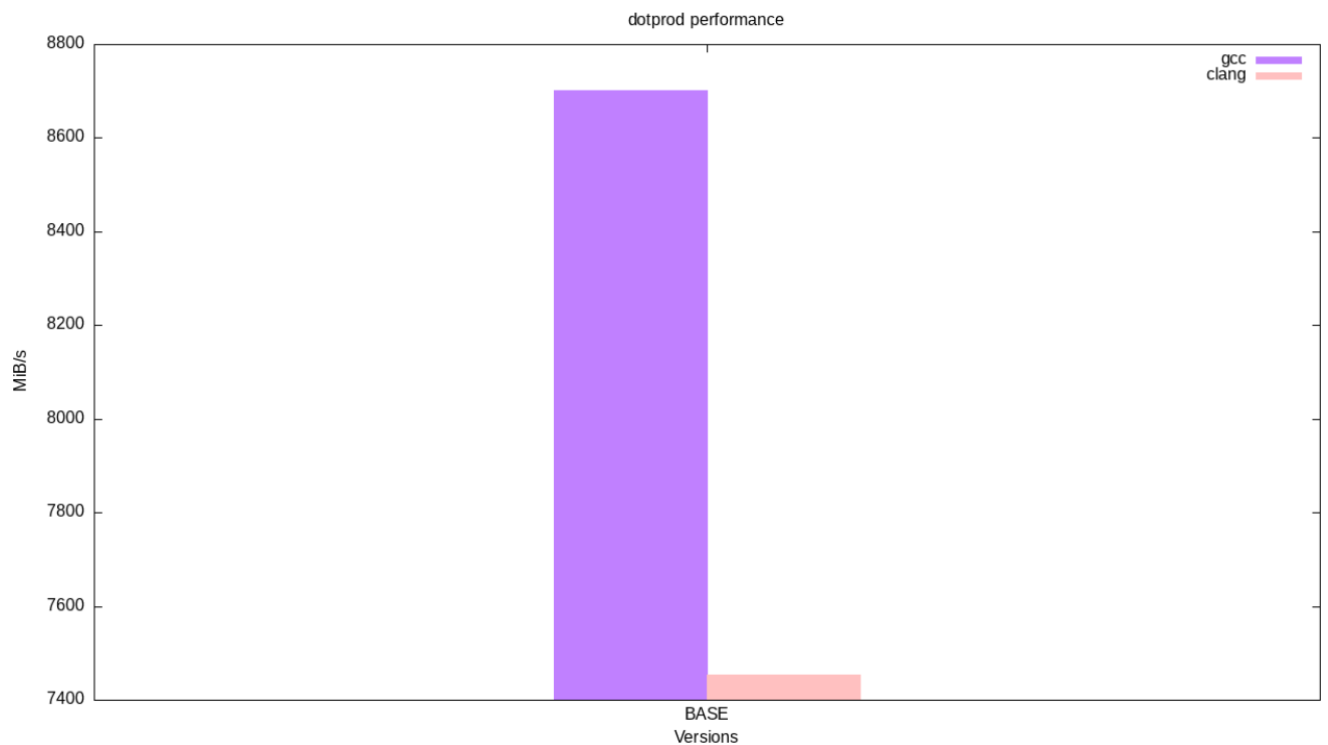


Figure 4: Figure qui illustre la différence entre les compilateurs pour dotprod.

La figure ci-dessous montre que le compilateur clang a donné de meilleurs résultats de performance que gcc pour reduc.

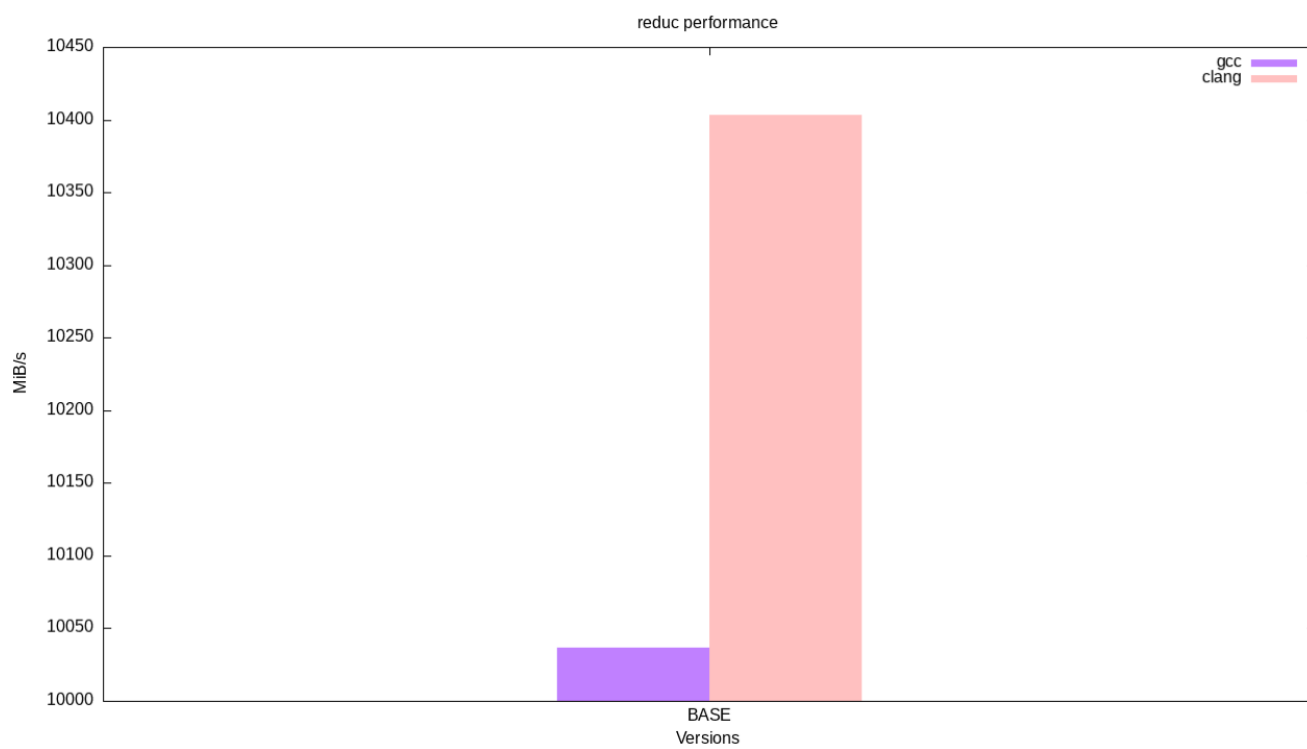


Figure 5: Figure qui illustre la différence entre les compilateurs pour reduc.

Les deux figures ci-dessous montrent que :

- La différence entre O0 et les autres niveaux d'optimisation est très importante pour les deux compilateurs
- O2 fournit les meilleures optimisations d'exécution pour clang.
- O3 fournit les meilleures optimisations d'exécution pour gcc.

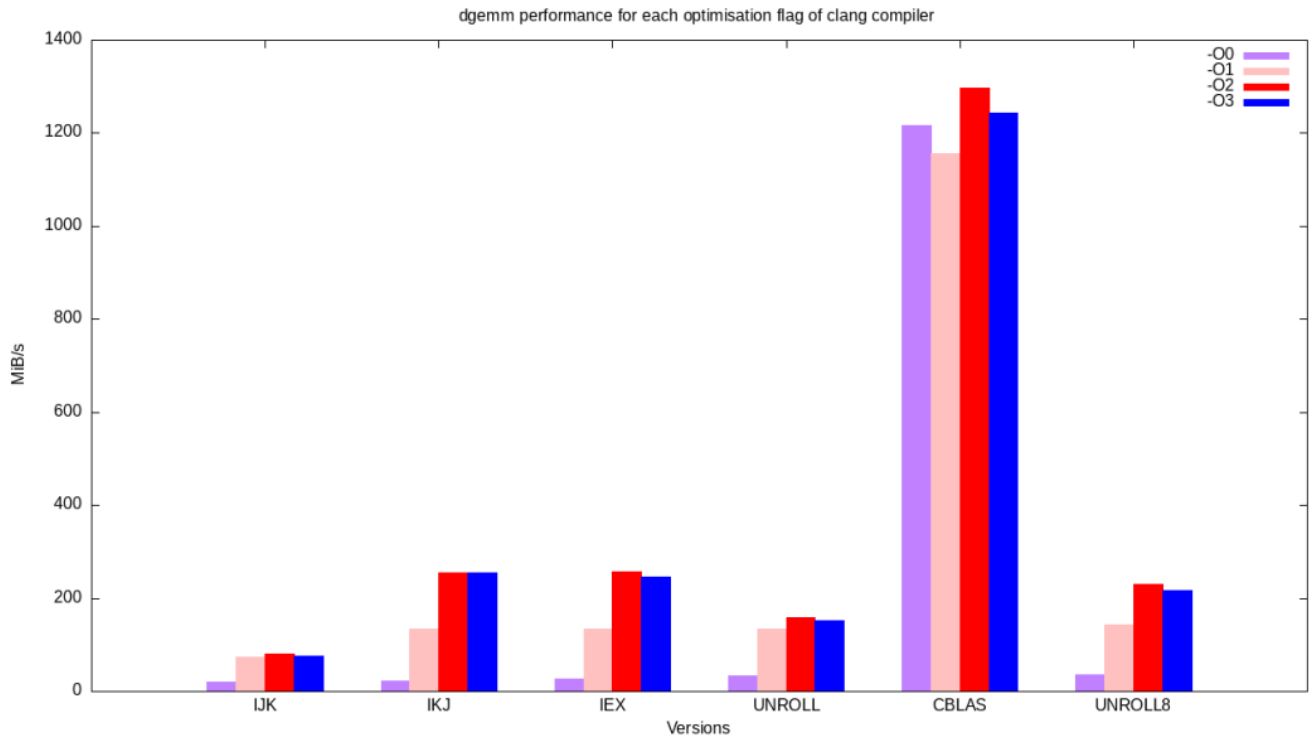


Figure 6: Figure qui illustre la différence entre les différents flags d'optimisation pour clang.

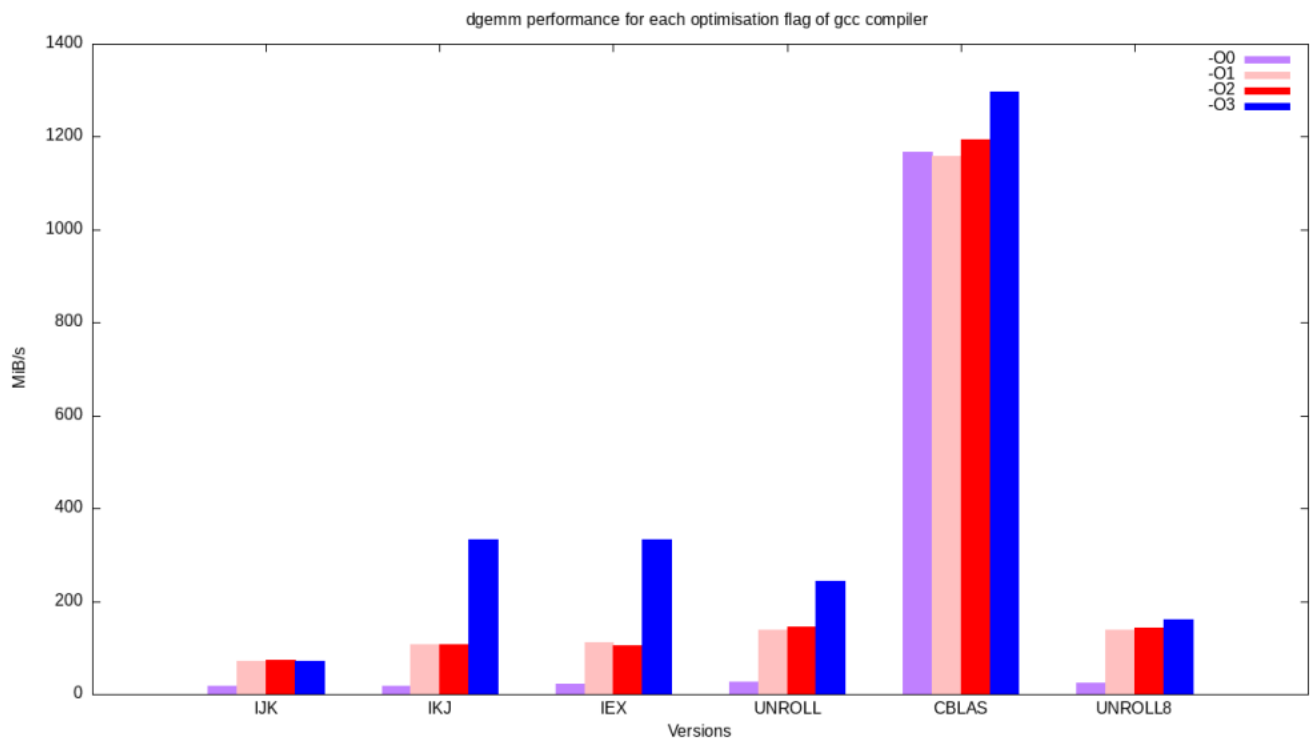


Figure 7: Figure qui illustre la différence entre les différents flags d'optimisation pour gcc.