UNIVERSITÉ CÔTE D'AZUR

MASTER INFORMATIQUE

Rapport de TP

Placement des threads sur les cœurs: Threads affinity

Réalisé par
Francois Flandin

Encadré par
Pr Sid Touati

Master Informatique
à Université Côte d'Azur
Premier semestre de l'annee 2024-2025

Table des matières

1 Introduction

Ce rapport s'inscrit dans le cadre des travaux pratiques visant à explorer les performances des programmes parallèles en étudiant l'impact du placement des threads sur les cœurs du processeur. L'objectif principal est de mesurer les variations de performances en fonction des différents paramètres d'exécution et de compiler des benchmarks pour analyser la stabilité des résultats. Pour cela, un code de multiplication de matrices massivement parallèle, implémenté avec OpenMP, a été utilisé comme point de départ. Différentes configurations ont été testées, en variant les stratégies de placement des threads sur les cœurs logiques. Les résultats obtenus ont été analysés à l'aide de techniques statistiques et de visualisations graphiques dans l'environnement R, afin de mettre en évidence les comportements de performance dans des contextes d'exécution variés. Ce document présente dans un premier temps la méthodologie employée, suivie d'une analyse des résultats expérimentaux, et conclut par une discussion sur les implications des choix de placement des threads sur l'efficacité des calculs parallèles.

Environnement Expérimental

Micro-Architecture

Dans cette partie sera detaillée la micro-architecture de la machine de tests.

Nom de modèle : 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
Taille des adresses : 39 bits physical, 48 bits virtual
Coeurs physiques : 4
Taille de ligne de cache : 64 octets

Graphical Topology				
Coeurs	0 4	1 5	2 6	3 7
Cache L1	48 kB	48 kB	48 kB	48 kB
Cache L2	1MB	1MB	1MB	1MB
Cache L3	8 MB			

Table 1 – Topologie de la machine de tests

Environnement Logiciel

Dans cette partie sera détaillée la partie logiciel de la machine de test, les fichiers scripts pour changer entre machine allégée et machine classique sont fournis dans l'archive.

Distribution : Fedora Linux v40 WorkStation
Compilateur utilisé : `gcc` version 14.2.1 20240912 avec option `-O2`
Outil de mesure des temps d'exécution : Commande `/bin/time`.

Configuration expérimentale

Processus en activité

```
> ps -x
PID TTY      STAT   TIME COMMAND
```

```
1263 ?        Ss     0:00 /usr/lib/systemd/systemd --user
1265 ?        S      0:00 (sd-pam)
1284 tty1     Ss     0:00 -zsh
1898 tty1     R+     0:00 ps -x
```

Description des processus

Voici ce que font les processus en cours d'exécution.

systemd : C'est un système d'initialisation et de gestion de services sous Linux, conçu pour démarrer, arrêter et superviser les processus, tout en offrant une gestion centralisée des sessions et des ressources.
sd-pam : *systemd-PAM* C'est un utilitaire de `systemd` pour gérer les sessions.
zsh : C'est le shell qui est lancé pour exécuter les commandes.
ps -x : C'est la commande qui a été lancée pour voir les processus.

Méthodologie de récolte des données expérimentales

Plusieurs scripts ont été réalisés afin de compiler le programme source sous différentes versions pour les besoins du TP, mais aussi afin de réaliser les différents benchmarks.
Pour mesurer le temps d'exécution, on utilisera la commande `/bin/time` sur 50 exécutions consécutives, ces données seront ensuite analysées à l'aide d'un programme `R`.

2 Compiler le code C, les differences entre `gcc` et `icx`

Dans cette partie, seront presentées les différences de librairies utilisées entre `gcc` et `icx` :

Librairies `matrix-icx`

```
> ldd matrix-icx

linux-vdso.so.1 (0x00007fca6e109000)
libm.so.6 => /lib64/libm.so.6 (0x00007fca6e004000)
libiomp5.so => /opt/intel/oneapi/2025.0/lib/libiomp5.so (0
    x00007fca6dc00000)
libc.so.6 => /lib64/libc.so.6 (0x00007fca6da0f000)
/lib64/ld-linux-x86-64.so.2 (0x00007fca6e10b000)
librt.so.1 => /lib64/librt.so.1 (0x00007fca6dfff000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007fca6dffa000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00007fca6da0a000)
```

Librairies `matrix-gcc`

```
> ldd matrix-gcc

linux-vdso.so.1 (0x00007fb525b31000)
libgomp.so.1 => /lib64/libgomp.so.1 (0x00007fb525aba000)
libc.so.6 => /lib64/libc.so.6 (0x00007fb5258c9000)
/lib64/ld-linux-x86-64.so.2 (0x00007fb525b33000)
```

Comparaison

On remarque que `icc` utilise bien plus de librairies que `gcc`, avec l'utilisation des librairies `libpthread`, `libdl` ou encore `librt`.
Cependant la plus grande différence est la librairie `OpenMP` qu'ils utilisent, à savoir `libiomp5` pour `icc` et `libgomp` pour `gcc`.

3 Stabilité des performances d'un programme

3.1 Résultat de `summary()` en R

matrix-gcc

```
> summary(X)
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
2.300   2.410   2.465   2.489   2.547   3.100
```

matrix-icx

```
> summary(Y)
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
5.420   5.643   5.770   5.828   5.987   6.720
```

On remarque que, d'une part `gcc` propose de meilleures performances que `icx` avec `OpenMP`, et d'autre part, que `gcc` propose des temps plus stables que `icx`, avec un écart-type de 0.8 secondes pour `gcc`, tandis que l'écart-type pour `icx` est de 1.3 secondes.

3.2 Boxplots

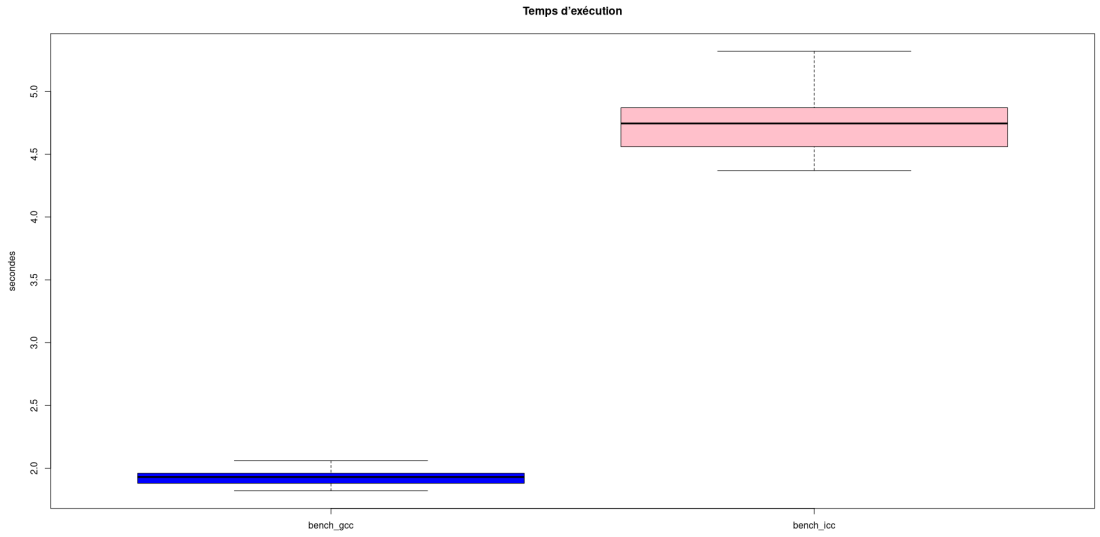

FIGURE 1 – Boxplots des temps d'exécution de matrix-gcc et matrix-gcc.

Analyse : Ce graphe montre que `gcc` est plus efficace que `icx` sans affinité de threads.

4 Analyse des stratégies `scatter` et `compact`

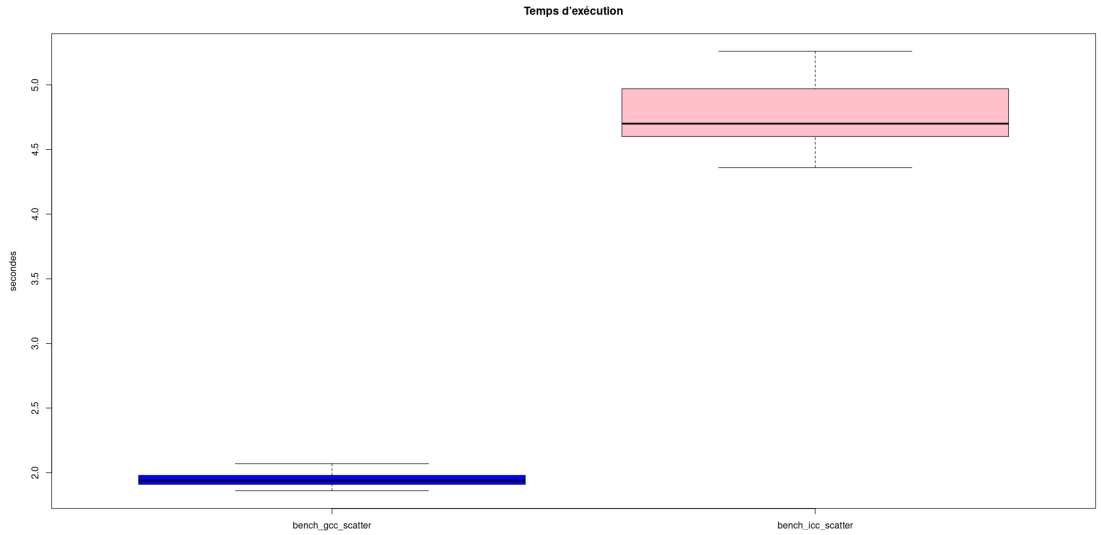

FIGURE 2 – Boxplots des temps d'execution de matrix-gcc et matrix-gcc avec l'affinite de threads `scatter`.

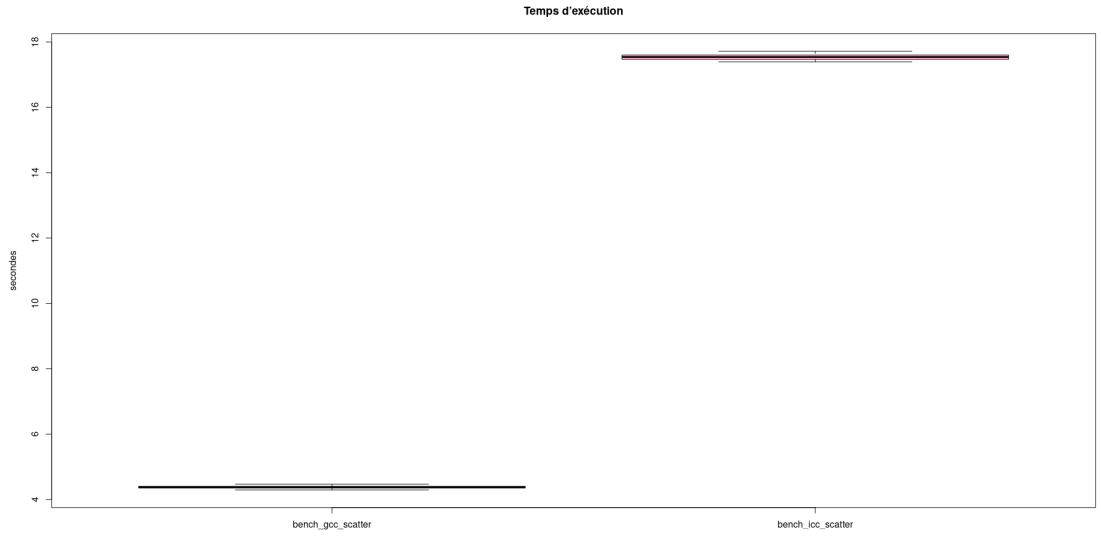

FIGURE 3 – Boxplots des temps d'execution de matrix-gcc et matrix-gcc avec l'affinite de threads `compact`.

5 Conclusion

Au cours de ces expériences, nous avons comparé la vitesse ainsi que la stabilité d'exécution de la libraire `OpenMP` avec `gcc` et `icx`, les résultats nous montrent que de manière générale, `gcc` produit des codes plus efficaces que `icx`.
De plus, des benchmarks sur d'autres stratégies d'affinité de thread ont été réalisés, montrant que la stratégie `scatter`, qui vise a séparer les threads sur lesquels on effectue l'exécution, est plus performante que la stratégie `compact`, qui elle vise a rapprocher les threads sur lesquel on effectue l'exécution. Cela peut être dû au fait que la stratégie `compact` utilise des coeurs logiques qui se trouvent sur le même coeur physique, ce qui pourrait diminuer l'efficacité par rapport à des threads sur des coeurs physiques séparés.