Analyse de la Parallélisation avec OpenMP

Marin Wimille et Céleste Koloussa

10 juin 2025

Résumé

Ce rapport présente une analyse approfondie de la parallélisation d'un produit matriciel en C utilisant OpenMP. La méthodologie repose sur l'arrêt de l'interface graphique (X11) afin de libérer toutes les ressources CPU, suivi du lancement d'un script d'expérimentation. Ce dernier exécute plusieurs tests en faisant varier le nombre de threads et la taille des tâches (chunk size) et stocke systématiquement les résultats dans un fichier CSV. Ces données pourront ensuite être exploitées via LibreOffice Calc ou un script Python pour générer des visualisations graphiques.

1 Introduction

Face à l'évolution des architectures multi-cœurs, la programmation parallèle s'impose comme une solution incontournable pour optimiser l'exécution des calculs intensifs. Dans cette étude, nous analysons l'impact de la parallélisation sur le calcul d'un produit matriciel en C, parallélisé avec OpenMP. L'objectif est d'étudier comment le nombre de threads et la taille des tâches influent sur les performances de l'application.

Pour garantir des mesures fiables, nous avons adopté une démarche en deux temps. Dans un premier temps, nous arrêtons l'interface graphique (X11) afin d'allouer l'ensemble des ressources CPU au benchmark. Dans un second temps, un script d'expérimentation lance une série de tests dont les résultats sont stockés dans un fichier CSV.

2 Méthodologie

La configuration expérimentale repose sur un programme C utilisant OpenMP et s'exécutant sur un processeur multi-cœurs. Les tests sont réalisés en variant le nombre de threads (1, 2, 4, 8, 16, 32, 64) ainsi que la taille des tâches (chunk size) allant de 1 à 1024. Chaque configuration est répétée cinq fois pour obtenir une moyenne des temps d'exécution, bien qu'un nombre de répétitions supérieur (50 à 100 itérations) aurait permis d'améliorer la précision.

Pour automatiser l'ensemble du processus, nous avons mis en place deux scripts Bash. Le premier, responsable de l'optimisation de l'interface, arrête le serveur X11 et lance le script d'expérimentation. Le second script réalise les tests et enregistre les résultats dans un fichier CSV.

3 Expérimentation et Collecte des Données

La mesure du temps d'exécution s'effectue à l'aide de la fonction `clock_gettime` dans le code C. Chaque configuration est testée cinq fois et le temps moyen est calculé. Voici un extrait du code utilisé pour la prise de mesures :

```
    for (int i = 0; i < 5; i++)
    {
        clock_gettime(CLOCK_REALTIME, &start);

        multiplyMatrix(&mat1, &mat2, &mat3, size, NB_THREADS);

        clock_gettime(CLOCK_REALTIME, &end);

        timespec_sub(&end, &start, &res);

        somme_sec += res.tv_sec;
        somme_nsec += res.tv_nsec;
    }
```

Listing 1 – Extrait de code pour la prise de mesure

Dans cette approche, la répétition de l'exécution permet d'atténuer les variations dues à d'éventuelles perturbations système. Bien qu'une répétition plus importante aurait permis d'affiner les résultats, cinq itérations offrent déjà une bonne première estimation des performances.

4 Analyse des Résultats

Les résultats expérimentaux permettent de mettre en lumière deux aspects importants du comportement du produit matriciel parallélisé avec OpenMP : l'influence de la taille des tâches sur le temps moyen d'exécution et l'impact du nombre de threads sur l'accélération.

4.1 Analyse du Temps Moyen d'Exécution en Fonction de la Taille des Tâches (chunk size)

La figure 1 présente le temps moyen d'exécution en fonction de la taille des tâches. On observe que :

- Pour des tailles de chunk très petites (par exemple 1, 2 ou 4), le temps d'exécution est relativement faible grâce à une bonne répartition du travail. Cependant, ce gain est contrebalancé par un surcoût de synchronisation, dû aux nombreuses attributions de tâches aux threads.
- Lorsque la taille des tâches augmente de manière intermédiaire (entre 16 et 64), on constate un compromis satisfaisant : le coût de synchronisation diminue, ce qui permet d'optimiser le temps d'exécution global.
- Pour des tailles trop importantes (256, 512, 1024), le déséquilibre entre les threads se traduit par une augmentation du temps d'exécution, certains threads se retrouvant sous-exploités tandis que d'autres sont surchargés.

Ces observations soulignent l'importance de choisir une taille de chunk adaptée afin d'équilibrer la charge de travail et de limiter les surcoûts de synchronisation. Dans notre

cas, la plage de 16 à 64 semble offrir les meilleures performances.

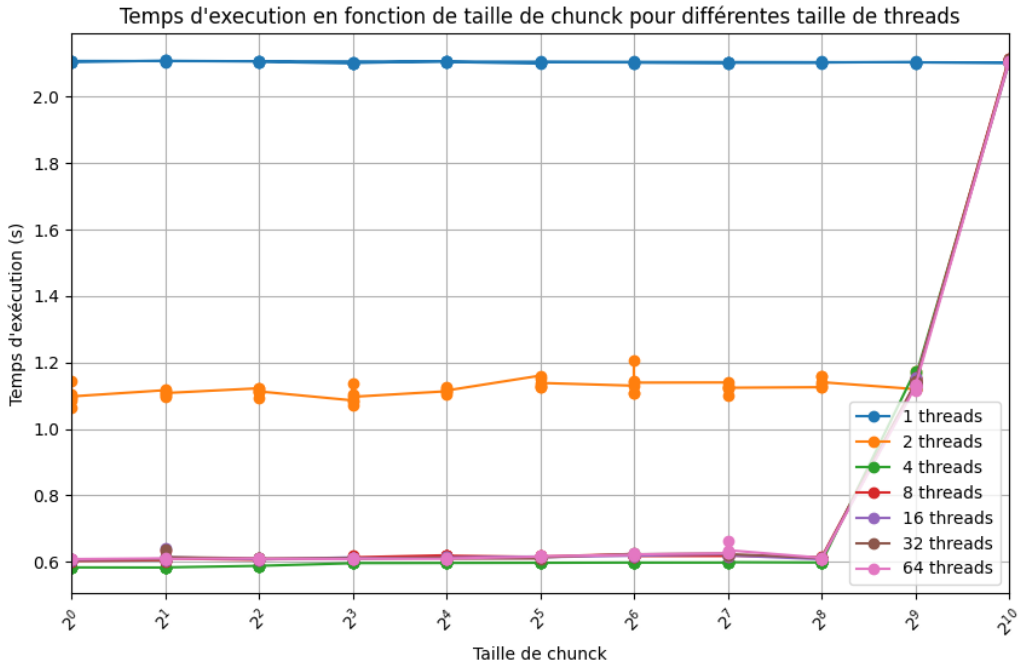

FIGURE 1 – Temps moyen d'exécution en fonction de la taille des tâches (chunk size)

4.2 Analyse de l'Accélération en Fonction du Nombre de Threads

La figure 2 illustre l'accélération obtenue en faisant varier le nombre de threads. L'analyse montre que :

- L'accélération est presque linéaire pour un nombre de threads faible, particulièrement jusqu'à 4 threads. Cela indique que la parallélisation permet d'exploiter efficacement plusieurs cœurs du processeur.
- Au-delà de 8 threads, on note une saturation des gains en accélération. En effet, l'augmentation du nombre de threads n'entraîne plus une réduction significative du temps d'exécution. Ce phénomène est principalement dû aux surcoûts liés à la synchronisation entre threads et aux limitations de la mémoire cache.
- Cette saturation suggère que, pour le produit matriciel testé, une utilisation optimale se situe entre 4 et 8 threads. Au-delà, les bénéfices de la parallélisation sont limités et la complexité de gestion du parallélisme devient un frein à l'amélioration des performances.

Ainsi, l'analyse des courbes d'accélération met en évidence l'importance de choisir un nombre de threads en adéquation avec l'architecture matérielle et la nature de la tâche à paralléliser.

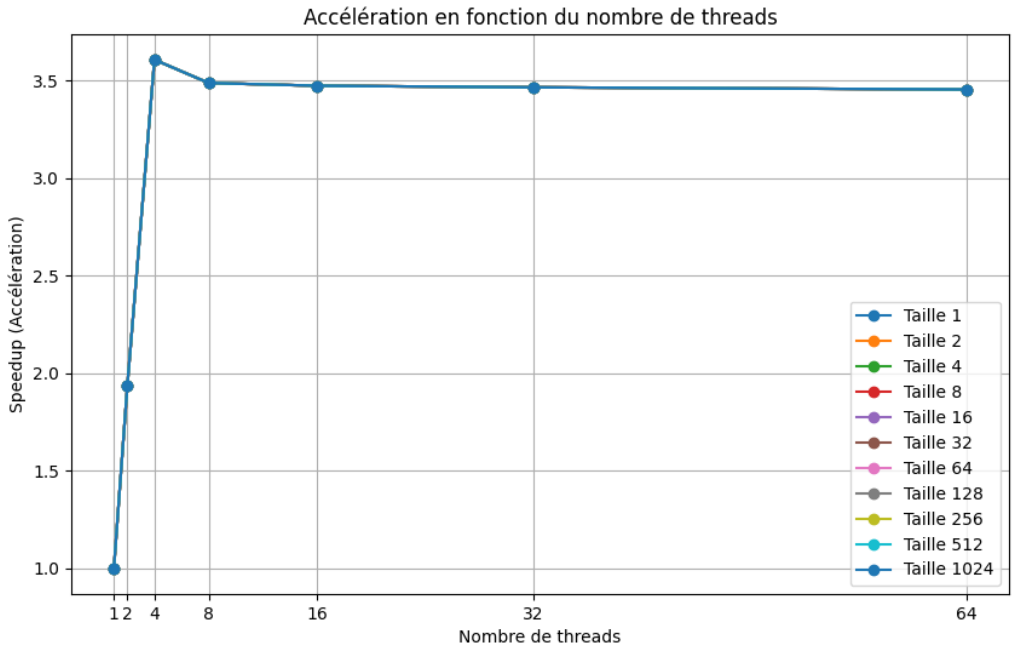

FIGURE 2 – Accélération en fonction du nombre de threads

5 Conclusion

L'analyse menée dans cette étude montre clairement que la parallélisation du produit matriciel avec OpenMP permet d'obtenir des améliorations de performances notables. L'utilisation judicieuse des threads permet en effet d'accélérer significativement le temps de calcul, avec des gains presque linéaires observés jusqu'à environ 4 threads et une accélération optimale jusqu'à 8 threads. Au-delà de ce seuil, les surcoûts liés à la synchronisation et les limitations de la mémoire cache entraînent une saturation des performances.

L'arrêt de l'interface graphique X11 a joué un rôle déterminant dans la fiabilité des mesures, en garantissant que l'intégralité des ressources CPU soit consacrée à l'exécution des benchmarks. Par ailleurs, l'automatisation de l'expérimentation via des scripts Bash, combinée au stockage des résultats dans un fichier CSV, offre une méthode reproductible et efficace pour analyser en détail l'impact de différents paramètres (nombre de threads et taille des tâches) sur les performances.

Ces résultats ouvrent la voie à de futures investigations, telles que la comparaison avec d'autres paradigmes de parallélisation, l'étude de l'impact sur des matrices de plus grande dimension ou encore l'évaluation sur des architectures matérielles variées. En somme, cette approche expérimentale fournit une base solide pour optimiser les applications parallèles et adapter les stratégies de parallélisation aux spécificités du matériel utilisé.

6 Annexes

Script d'expérimentation et stockage des résultats

Le script d'expérimentation parcourt l'ensemble des configurations en faisant varier le nombre de threads et la taille des tâches. Pour chaque configuration, le temps moyen d'exécution est mesuré et ajouté au fichier CSV :

```
#!/bin/bash

# Fichier o  stocker les r sultats
OUTPUT_FILE="results.csv"

# Nombre de r p titions pour fiabiliser les mesures
REPEATS=5

# Liste des tailles de t ches OpenMP    tester
SIZES=(1 2 4 8 16 32 64 128 256 512 1024)

# Liste des nombres de threads    tester
THREADS=(1 2 4 8 16 32 64)

# V rifier si le fichier existe, sinon ajouter l'en-t te
if [ ! -f "$OUTPUT_FILE" ]; then
    echo "size,num_threads,mean_time" > $OUTPUT_FILE
fi

# Boucle sur les tailles de t ches et le nombre de threads
for NB_THREADS in "${THREADS[@]}"
do
    for SIZE in "${SIZES[@]}"
    do
        echo "Running␣with␣size=$SIZE␣and␣threads=$NB_THREADS..."

        SUM_TIME=0.0

        for (( i=0; i<$REPEATS; i++ ))
        do
            # Ex cuter le programme et r cup rer la sortie
               proprement
            OUTPUT=$(sudo ./out $SIZE $NB_THREADS)

            # Extraire la valeur du temps
            TIME=$(echo "$OUTPUT" | cut -d',' -f3)

            # V rifier que la valeur est bien un nombre
            if [[ "$TIME" =~ ^[0-9]+(\.[0-9]+)?$ ]]; then
                SUM_TIME=$(echo "$SUM_TIME␣+␣$TIME" | bc)
            else
                echo "Erreur␣:␣Valeur␣non␣num rique␣d tect e␣:␣
                   $TIME"
                exit 1
```

```
            fi
        done

        # Calculer la moyenne avec bc
        MEAN_TIME=$(echo "scale=6; $SUM_TIME / $REPEATS" | bc)

        # Enregistrer dans le fichier CSV
        echo "$SIZE,$NB_THREADS,$MEAN_TIME" >> $OUTPUT_FILE
    done
done

echo "Tests termin s , r sultats enregistr s dans $OUTPUT_FILE."
#!/bin/bash

# Fichier o  stocker les r sultats
OUTPUT_FILE="results.csv"

# Nombre de r p titions pour fiabiliser les mesures
REPEATS=5

# Liste des tailles de t ches OpenMP    tester
SIZES=(1 2 4 8 16 32 64 128 256 512 1024)

# Liste des nombres de threads    tester
THREADS=(1 2 4 8 16 32 64)

# V rifier si le fichier existe, sinon ajouter l'en-t te
if [ ! -f "$OUTPUT_FILE" ]; then
    echo "size,num_threads,mean_time" > $OUTPUT_FILE
fi

# Boucle sur les tailles de t ches et le nombre de threads
for NB_THREADS in "${THREADS[@]}"
do
    for SIZE in "${SIZES[@]}"
    do
        echo "Running with size=$SIZE and threads=$NB_THREADS..."

        SUM_TIME=0.0

        for (( i=0; i<$REPEATS; i++ ))
        do
            # Ex cuter le programme et r cup rer la sortie
               proprement
            OUTPUT=$(sudo ./out $SIZE $NB_THREADS)

            # Extraire la valeur du temps
            TIME=$(echo "$OUTPUT" | cut -d',' -f3)

            # V rifier que la valeur est bien un nombre
            if [[ "$TIME" =~ ^[0-9]+(\.[0-9]+)?$ ]]; then
```

```
93                  SUM_TIME=$(echo "$SUM_TIME␣+␣$TIME" | bc)
94              else
95                  echo "Erreur␣:␣Valeur␣non␣num rique␣d tect e␣:␣
                        $TIME"
96                  exit 1
97              fi
98          done
99
100         # Calculer la moyenne avec bc
101         MEAN_TIME=$(echo "scale=6;␣$SUM_TIME␣/␣$REPEATS" | bc)
102
103         # Enregistrer dans le fichier CSV
104         echo "$SIZE,$NB_THREADS,$MEAN_TIME" >> $OUTPUT_FILE
105     done
106 done
107
108 echo "Tests␣termin s ,␣r sultats␣enregistr s␣dans␣$OUTPUT_FILE."
109 #!/bin/bash
110
111 # Fichier o  stocker les r sultats
112 OUTPUT_FILE="results.csv"
113
114 # Nombre de r p titions pour fiabiliser les mesures
115 REPEATS=5
116
117 # Liste des tailles de t ches OpenMP    tester
118 SIZES=(1 2 4 8 16 32 64 128 256 512 1024)
119
120 # Liste des nombres de threads    tester
121 THREADS=(1 2 4 8 16 32 64)
122
123 # V rifier si le fichier existe, sinon ajouter l'en-t te
124 if [ ! -f "$OUTPUT_FILE" ]; then
125     echo "size,num_threads,mean_time" > $OUTPUT_FILE
126 fi
127
128 # Boucle sur les tailles de t ches et le nombre de threads
129 for NB_THREADS in "${THREADS[@]}"
130 do
131     for SIZE in "${SIZES[@]}"
132     do
133         echo "Running␣with␣size=$SIZE␣and␣threads=$NB_THREADS..."
134
135         SUM_TIME=0.0
136
137         for (( i=0; i<$REPEATS; i++ ))
138         do
139             # Ex cuter le programme et r cup rer la sortie
                    proprement
140             OUTPUT=$(sudo ./out $SIZE $NB_THREADS)
141
```

```
142             # Extraire la valeur du temps
143             TIME=$(echo "$OUTPUT" | cut -d',' -f3)
144 
145             # Vérifier que la valeur est bien un nombre
146             if [[ "$TIME" =~ ^[0-9]+(\.[0-9]+)?$ ]]; then
147                 SUM_TIME=$(echo "$SUM_TIME + $TIME" | bc)
148             else
149                 echo "Erreur : Valeur non numérique détectée : $TIME"
150                 exit 1
151             fi
152         done
153 
154         # Calculer la moyenne avec bc
155         MEAN_TIME=$(echo "scale=6; $SUM_TIME / $REPEATS" | bc)
156 
157         # Enregistrer dans le fichier CSV
158         echo "$SIZE,$NB_THREADS,$MEAN_TIME" >> $OUTPUT_FILE
159     done
160 done
161 
162 echo "Tests terminés, résultats enregistrés dans $OUTPUT_FILE."
```

Listing 2 – Script d'expérimentation et stockage des résultats dans un fichier CSV

Script d'optimisation et lancement du benchmark

Le script suivant arrête l'interface graphique et exécute le benchmark :

```
#!/bin/bash

# V rifier si le script est ex cut  en root
if [ "$(id -u)" -ne 0 ]; then
    echo "Ce script doit tre ex cut en tant que root. Utilise sudo."
    exit 1
fi

# Nom du fichier CSV des r sultats
OUTPUT_FILE="results.csv"

# Arr ter l'interface graphique pour lib rer les ressources
echo "Arr t de l'interface graphique..."
sudo systemctl stop gdm 2>/dev/null || sudo systemctl stop sddm 2>/dev/null || sudo systemctl stop lightdm 2>/dev/null

# Attendre 2 secondes pour s'assurer que tout est arr t
sleep 2

# V rifier si l'environnement graphique est bien arr t
if pgrep Xorg > /dev/null || pgrep gnome-shell > /dev/null; then
    echo "Erreur : L'environnement graphique est toujours actif. Arr t manuel requis."
    exit 1
fi

echo "Toutes les ressources CPU sont maintenant disponibles."

# Ex cuter le benchmark
echo "Ex cution du benchmark..."
chmod +x benchmark.sh
sudo ./benchmark.sh

# V rifier si les r sultats ont  t  g n r s
if [ -f "$OUTPUT_FILE" ]; then
    echo "Benchmark termin avec succ s ! R sultats enregistr s dans $OUTPUT_FILE."
else
    echo "Erreur : Le benchmark ne semble pas avoir produit de r sultats."
fi

# Red marrer l'interface graphique
echo "Red marrage de l'interface graphique..."
sudo systemctl start gdm 2>/dev/null || sudo systemctl start sddm 2>/dev/null || sudo systemctl start lightdm 2>/dev/null

```

```
43 echo "Tout␣est␣revenu␣  ␣la␣normale."
44 #!/bin/bash
45
46 # V rifier si le script est ex cut  en root
47 if [ "$(id␣-u)" -ne 0 ]; then
48     echo "Ce␣script␣doit␣ tre ␣ex cut ␣en␣tant␣que␣root.␣Utilise␣
       sudo."
49     exit 1
50 fi
51
52 # Nom du fichier CSV des r sultats
53 OUTPUT_FILE="results.csv"
54
55 # Arr ter l'interface graphique pour lib rer les ressources
56 echo "Arr t␣de␣l'interface␣graphique..."
57 sudo systemctl stop gdm 2>/dev/null || sudo systemctl stop sddm 2>/
   dev/null || sudo systemctl stop lightdm 2>/dev/null
58
59 # Attendre 2 secondes pour s'assurer que tout est arr t
60 sleep 2
61
62 # V rifier si l'environnement graphique est bien arr t
63 if pgrep Xorg > /dev/null || pgrep gnome-shell > /dev/null; then
64     echo "Erreur␣:␣L'environnement␣graphique␣est␣toujours␣actif.␣
       Arr t␣manuel␣requis."
65     exit 1
66 fi
67
68 echo "Toutes␣les␣ressources␣CPU␣sont␣maintenant␣disponibles."
69
70 # Ex cuter le benchmark
71 echo "Ex cution␣du␣benchmark..."
72 chmod +x benchmark.sh
73 sudo ./benchmark.sh
74
75 # V rifier si les r sultats ont  t  g n r s
76 if [ -f "$OUTPUT_FILE" ]; then
77     echo "Benchmark␣termin ␣avec␣succ s␣!␣R sultats␣enregistr s
       ␣dans␣$OUTPUT_FILE."
78 else
79     echo "Erreur␣:␣Le␣benchmark␣ne␣semble␣pas␣avoir␣produit␣de␣
       r sultats."
80 fi
81
82 # Red marrer l'interface graphique
83 echo "Red marrage␣de␣l'interface␣graphique..."
84 sudo systemctl start gdm 2>/dev/null || sudo systemctl start sddm
   2>/dev/null || sudo systemctl start lightdm 2>/dev/null
85
86 echo "Tout␣est␣revenu␣  ␣la␣normale."
```

Listing 3 – Script d'optimisation de l'interface et lancement du benchmark

Script Python pour la visualisation des résultats

Pour analyser les données, le fichier CSV généré peut être importé dans LibreOffice Calc ou traité par un script Python. Par exemple, le script suivant utilise les bibliothèques pandas et matplotlib pour tracer l'accélération en fonction du nombre de threads pour une taille de chunk donnée :

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Charger les donn es
df = pd.read_csv("results.csv")

# Utiliser directement la colonne mean_time
df["time"] = df["mean_time"]

# Graphique 1 : Temps d'ex cution en fonction de la taille des
    t ches
plt.figure(figsize=(10, 6))
for num_threads in df["num_threads"].unique():
    subset = df[df["num_threads"] == num_threads]
    plt.plot(subset["size"], subset["time"], marker="o", linestyle=
        "-", label=f"{num_threads} threads")

plt.xlabel("Taille de chunck")
plt.ylabel("Temps d'ex cution (s)")
plt.xscale("log", base=2)
plt.xlim(1, 1024)
# plt.yscale("log")
plt.title("Temps d'execution en fonction de taille de chunck pour
    diff rentes taille de threads")
plt.legend()
plt.grid()

# R duction du pas de l'axe X pour plus de d tails
plt.xticks(df["size"].unique(), rotation=45)  # Affichage des
    tailles exactes
plt.savefig("performance_vs_size.png")
plt.show()

# Graphique 2 : Speedup en fonction du nombre de threads
df_base = df[df["num_threads"] == 1]
speedup = {}

for num_threads in df["num_threads"].unique():
    df_threads = df[df["num_threads"] == num_threads]

    # Correction : s'assurer que les tableaux ont la m me taille
    min_size = min(len(df_base), len(df_threads))
    speedup[num_threads] = df_base["time"].values[:min_size] /
        df_threads["time"].values[:min_size]
```

```
41
42 plt.figure(figsize=(10, 6))
43 for size in df["size"].unique():
44     plt.plot(df["num_threads"].unique(), [speedup[nt][i] for i, nt
       in enumerate(df["num_threads"].unique()) if nt in speedup],
       marker="o", linestyle="-", label=f"Taille␣{size}")
45
46 plt.xlabel("Nombre␣de␣threads")
47 plt.ylabel("Speedup␣(Acc l ration)")
48 plt.title("Acc l ration␣en␣fonction␣du␣nombre␣de␣threads")
49 plt.legend()
50 plt.grid()
51
52 # R duction du pas de l'axe X
53 plt.xticks(df["num_threads"].unique())  # Affichage d taill  du
   nombre de threads
54
55 plt.savefig("speedup_vs_threads.png")
56 plt.show()
```

Listing 4 – Script Python pour générer un graphique