TP : EXPÉRIMENTATION DU PRODUIT MATRICIEL AVEC OPENMP

MEMBRES

NESRINE KHIARI

HOUCEM HBIRI

كلية العلوم بتونس

FACULTE DES SCIENCES DE TUNIS

INTRODUCTION: OBJECTIFS ET MÉTHODOLOGIE

L’objectif principal de ce projet est d’analyser l’impact de la parallélisation sur les performances d’un produit matriciel en utilisant le langage C et OpenMP. En exploitant différentes tailles de matrices, stratégies de parallélisation et configurations matérielles (nombre de cœurs), nous visons à étudier comment la parallélisation influence le temps d’exécution (makespan) et le gain de performance (speedup) par rapport à une exécution séquentielle.

La méthodologie suivie s'articule autour des étapes suivantes :

1. **Implémentation Séquentielle :** Une version classique du produit matriciel a été développée pour servir de référence de performance.
2. **Parallélisation avec OpenMP :** Plusieurs variantes ont été conçues en utilisant les directives OpenMP pour paralléliser différentes boucles imbriquées.
3. **Évaluation des Stratégies :** Les versions parallèles ont été testées avec des stratégies de planification telles que statique, dynamique et par blocs, tout en variant le nombre de cœurs de 1 à 8.
4. **Analyse des Performances :** Les mesures collectées incluent le temps d’exécution moyen sur 5 essais, et l’accélération a été calculée pour chaque configuration afin de quantifier les gains de parallélisation. Les résultats ont été analysés et présentés sous forme de tableaux et de graphiques.

Cette approche permet de comprendre les bénéfices et limitations de la parallélisation en fonction des paramètres choisis, et de tirer des conclusions sur l’efficacité d’OpenMP pour ce type de problème

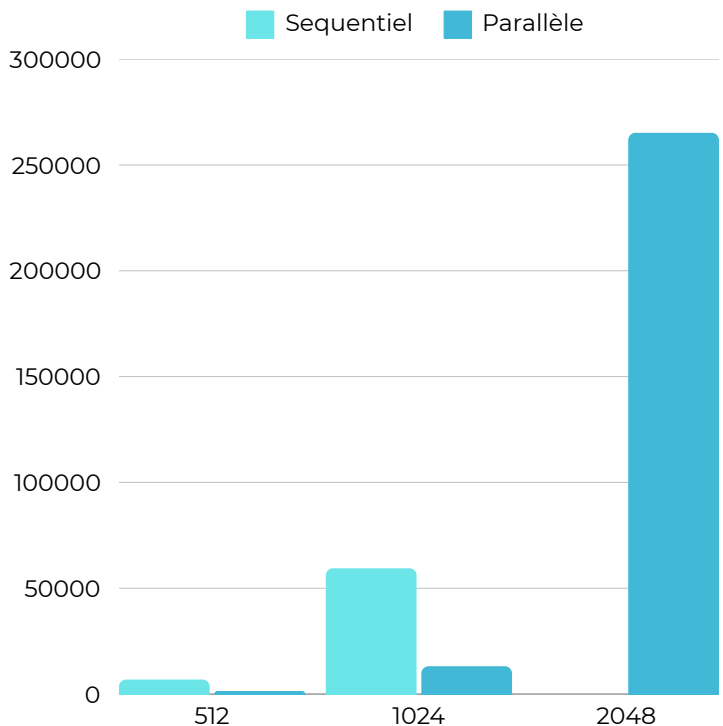
REPRESENTATION DES RESULTATS ET ANALYSE

1. **Temps séquentiel vs. meilleur temps parallèle par taille**

Taille d'entrée	Temps séquentiel (s)	Meilleur temps Parallèle(s)	Gain (%)
512x512	0.6842	0.1462 (Guided, 8 threads)	78.63
1024x1024	5.9422	1.3138 (Guided, 4 threads)	77.89
2048x2048	111.1106	26.5186 (Static, 8 threads)	76.13

Analyse :

1. **Réduction du temps d'exécution :**
   - Les temps parallèles montrent une amélioration significative par rapport au temps séquentiel, confirmant l'efficacité de la parallélisation.
   - Pour les tailles plus grandes (ex : 2048x2048), la différence est plus prononcée, démontrant l'importance de la parallélisation pour des charges de travail élevées.
2. **Stratégie gagnante :**
   - La stratégie "Guided" domine pour les deux premières tailles d'entrée. Cela s'explique par la flexibilité qu'elle offre, surtout lorsque les tailles de blocs diminuent dynamiquement.
   - Pour la taille 2048x2048, la stratégie "Static" est utilisée avec 8 threads. Cela pourrait s'expliquer par une meilleure uniformité dans la charge de travail puisque les tâches sont similaires et distribuées de manière équilibrée.



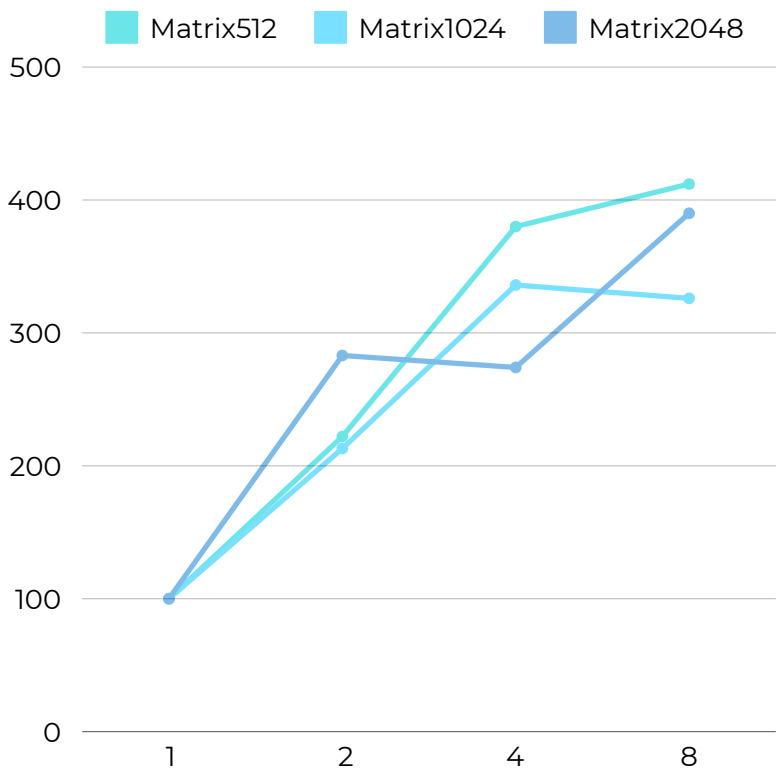
2. Moyennes de speedup par stratégie et nombre de threads

Stratégie	2 Threads	4 Threads	8 Threads
Static	2.07	3.63	4.12
Dynamic	2.26	3.90	4.09
Guided	2.32	3.88	4.16

Analyse :

1. **Tendance générale :**
   - Le speedup augmente avec le nombre de threads, ce qui est attendu puisque plus de threads permettent une parallélisation accrue.
   - Cependant, les gains diminuent à mesure que le nombre de threads augmente, ce qui indique une saturation des ressources ou des coûts supplémentaires liés à la gestion des threads (overhead).
2. **Stratégies comparées :**
   - Static :
     - Performante pour des tâches bien équilibrées entre threads, mais moins efficace si les charges de travail varient.
     - Donne des résultats stables avec une augmentation progressive du speedup.
   - Dynamic :
     - Mieux adaptée aux charges de travail inégales, ce qui explique des moyennes légèrement supérieures pour 2 et 4 threads.
     - Une fois les threads saturés (8 threads), les avantages par rapport à Static s'estompent.
   - Par Bloc:
     - Offre une bonne performance initiale avec un faible nombre de threads, en particulier pour 2 et 4 threads.
     - Les performances tendent à égaler celles de Dynamic avec 8 threads.
3. **Nombre optimal de threads :**
   - Pour les tailles d'entrée analysées, 4 threads semblent offrir un bon équilibre entre parallélisation et overhead.

3. Evolution de la performance (accélération) en fct du nombre de threads



Analyse

1. **Tendance générale :** On observe un gain de performance à mesure que le nombre de threads augmente, mais ce gain n'est pas toujours linéaire.

- Pour les petites matrices (512x512), le speedup plafonne légèrement entre 4 et 8 threads (3.80 → 4.12).
- Pour les grandes matrices (2048x2048), le gain est plus significatif, avec une meilleure exploitation des threads supplémentaires.

2. **Effet de la parallélisation :**

- Lorsque le nombre de threads double (par exemple, de 2 à 4), le speedup ne double pas systématiquement. Cela est dû à des facteurs comme la surcharge de gestion des threads et la limitation des ressources matérielles.

3. **Impact de la taille des matrices**

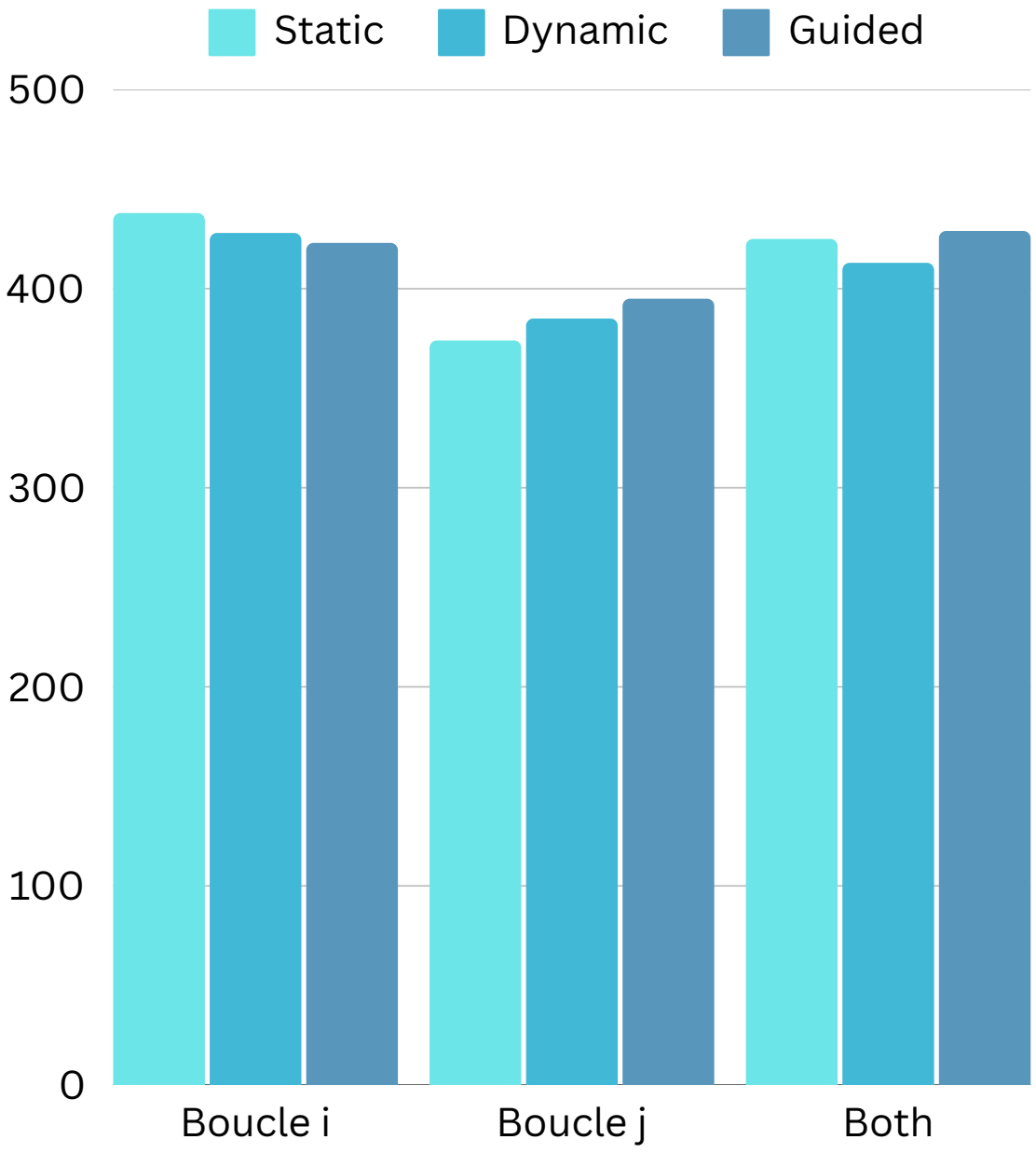
- **Comportement non linéaire :** Dans certaines configurations (par exemple, 1024x1024 avec 8 threads), le speedup diminue légèrement. Cela peut être attribué à des phénomènes comme la contention mémoire ou l'augmentation des communications entre threads.ads :

Optimisation possible :

Pour exploiter pleinement les threads supplémentaires, des optimisations comme la réduction de la contention mémoire ou une meilleure répartition des tâches peuvent être nécessaires.

4. Comparaison des niveaux pour chaque stratégie

Stratégie	Niveau 1 (speedup)	Niveau 2 (Speedup)	Niveau 3 (Speedup)	Moyenne
Static	4.38	3.74	4.25	4.12
Dynamic	4.28	3.85	4.13	4.09
Guided	4.23	3.95	4.29	4.16



Analyse:

1. Performances globales (Moyenne des stratégies) :

- Guided offre la meilleure performance moyenne (4.16) grâce à sa flexibilité, ce qui est particulièrement efficace pour des charges de travail déséquilibrées ou dynamiques.

2. Analyse par niveau de parallélisation :

- Niveau 1 (parallélisation de i) :
  - Static est la meilleure stratégie avec un speedup de 4.38. Cela s'explique par la charge de travail bien équilibrée lorsque la boucle extérieure est parallélisée, les itérations étant uniformes.
  - Dynamic (4.28) et Guided (4.23) restent proches, mais leur flexibilité est moins critique ici puisque chaque itération de la boucle i effectue une quantité similaire de calculs.
- Niveau 2 (parallélisation de j) :
  - Guided est supérieure avec un speedup de 3.95, grâce à une meilleure répartition des tâches lorsque la boucle intermédiaire est parallélisée. Cela est utile si les lignes de la matrice ou les colonnes contiennent des variations de charge.
- Niveau 3 (parallélisation combinée de i et j) :
  - Guided se démarque avec un speedup de 4.29, gérant efficacement la complexité accrue de la parallélisation sur deux niveaux. Cela est utile lorsque les tailles des blocs à traiter varient.

CONCLUSION : SYNTHÈSE DES OBSERVATIONS ET RECOMMANDATIONS

1. **Réduction du temps d'exécution**
   - La parallélisation offre des gains significatifs, particulièrement pour les matrices de grande taille (ex. 2048x2048).
   - L'efficacité des stratégies dépend de la taille des données et de la nature des charges de travail.
2. **Tendance générale du speedup**
   - L'augmentation du nombre de threads entraîne un speedup notable, mais avec des rendements décroissants au-delà d'un certain seuil, en raison de la saturation des ressources et des surcoûts liés à la gestion des threads.
3. **Recommandations pour optimisation**
   - Nombre optimal de threads : Pour les tailles de matrices analysées, 4 threads offrent souvent le meilleur compromis entre performance et overhead.
   - Optimisations à envisager :
     - Réduction de la contention mémoire en optimisant les accès aux données.
     - Une meilleure répartition des tâches, notamment via des algorithmes de partitionnement adaptés.
     - Une stratégie dynamique ou guidée pour des tâches déséquilibrées ou lorsque la parallélisation est combinée sur plusieurs niveaux.
4. **Recommandations stratégiques**
   - Privilégier Static pour des charges équilibrées (ex. boucle extérieure parallélisée).
   - Adopter Guided pour des scénarios complexes ou des tailles de blocs variables.
   - Envisager Dynamic pour des charges inégales, notamment sur des matrices de taille moyenne ou petite.

En résumé, la parallélisation, bien que très efficace, nécessite une sélection stratégique des approches en fonction des configurations spécifiques. Un équilibrage judicieux des threads et des stratégies permet de maximiser les gains tout en minimisant les surcoûts.