Aroua Rahma
Belhadj Ghada
IGL4

Expérimentation du produit matriciel avec OpenMP

Matière: Calcul parallèle

Partie 1 : Génération et multiplication de matrices

1.allocation et génération des matrices

```c
// Partie 1 : Allocation et génération des matrices
float** allocate_matrix(int size) {
    float** matrix = (float**)malloc(size * sizeof(float*));
    for (int i = 0; i < size; i++) {
        matrix[i] = (float*)malloc(size * sizeof(float));
    }
    return matrix;
}

void free_matrix(float** matrix, int size) {
    for (int i = 0; i < size; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

void generate_matrix(float** matrix, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            matrix[i][j] = (float)(rand() % 100);
        }
    }
}
```

2.Produit matriciel séquentiel :

```
// Produit matriciel séquentiel
void matrix_multiply_sequential(float** A, float** B, float** C, int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            C[i][j] = 0;
            for (int k = 0; k < size; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

Partie 2 : Parallélisation avec OpenMP

Résumé :

1. **Versions parallèles**
    - On a implémenté trois versions parallèles avec les directives `#pragma omp for` :
        - Parallélisation de la boucle **i**.
        - Parallélisation de la boucle **j**.
        - Parallélisation simultanée des boucles **i** et **j**.
2. **Stratégies de parallélisation**
    - On a testé ces versions avec différentes stratégies d'OpenMP :
        - **Statique** : Répartition fixe des itérations entre les threads.
        - **Dynamique** : Répartition au fur et à mesure de la disponibilité des threads.
        - **Par blocs** : Découpage des itérations en blocs prédéfinis.
3. **Configurations expérimentales**
    - On a testé avec différentes tailles de matrices.

- On a fait varier le nombre de cœurs de 1 à 8.

On peut consulter les détails des fonctions dans le fichier source.

Partie 3 : Analyse des performances :

1. Taille de matrice : 512x512

Nombre de threads	Stratégie	Temps (secondes)	Speedup
1	Statique	0.541468	1.481358
1	Dynamique	0.539889	1.485692
1	Par blocs	0.537133	1.493314
2	Statique	0.545978	1.469124
2	Dynamique	0.532022	1.507662
2	Par blocs	0.541520	1.481218
4	Statique	0.592708	1.353294
4	Dynamique	0.560750	1.430421
4	Par blocs	0.547804	1.464226
8	Statique	0.517465	1.550072
8	Dynamique	0.552705	1.451241
8	Par blocs	0.578601	1.386290

2. Taille de matrice : 1024x1024 :

Nombre de threads	Stratégie	Temps (secondes)	Speedup
1	Statique	8.244665	1.023292
1	Dynamique	7.883330	1.070195
1	Par blocs	8.021963	1.051701
2	Statique	7.888849	1.069447
2	Dynamique	7.763590	1.086701
2	Par blocs	7.657876	1.101703
4	Statique	7.629343	1.105823
4	Dynamique	7.750442	1.088545
4	Par blocs	7.895242	1.068581
8	Statique	7.690167	1.097077
8	Dynamique	11.302524	0.746444
8	Par blocs	18.474206	0.456675

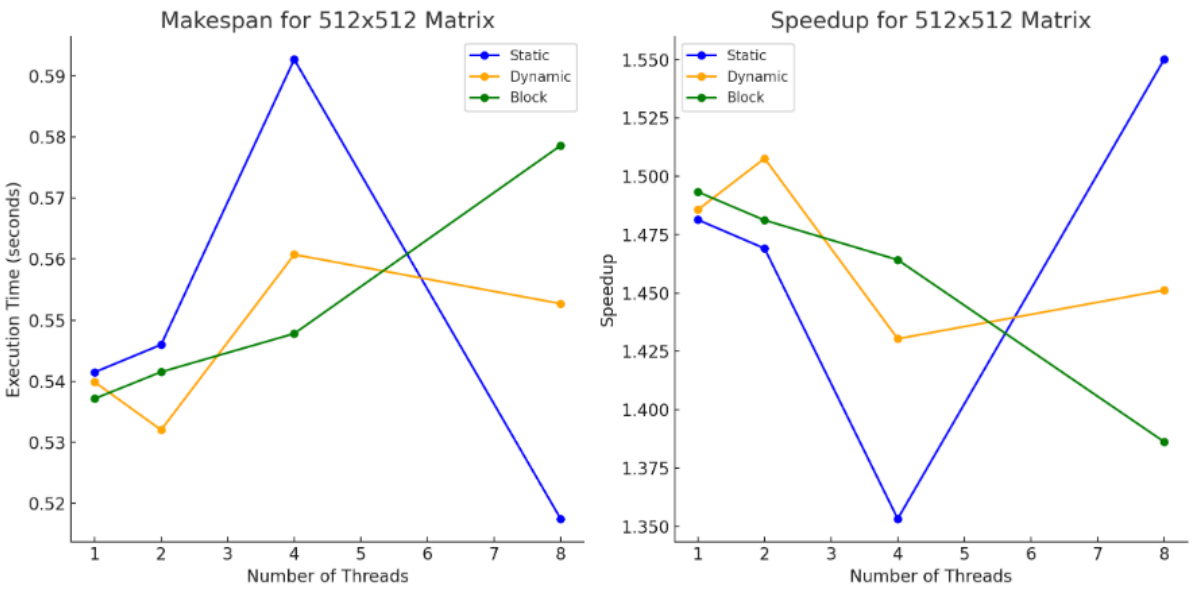
3. Taille de matrice : 2048x2048 :

Nombre de threads	Stratégie	Temps (secondes)	Speedup
1	Statique	92.630309	1.555337
1	Dynamique	125.393282	1.148956
1	Par blocs	154.094133	0.934957
2	Statique	171.611281	0.839521
2	Dynamique	195.050351	0.738637
2	Par blocs	195.021984	0.738744
4	Statique	195.235006	0.737938
4	Dynamique	194.796031	0.739601
4	Par blocs	1074.821328	0.134042
8	Statique	146.217103	0.985325
8	Dynamique	97.969629	1.470571
8	Par blocs	99.831898	1.443139

Les courbes comparatives :

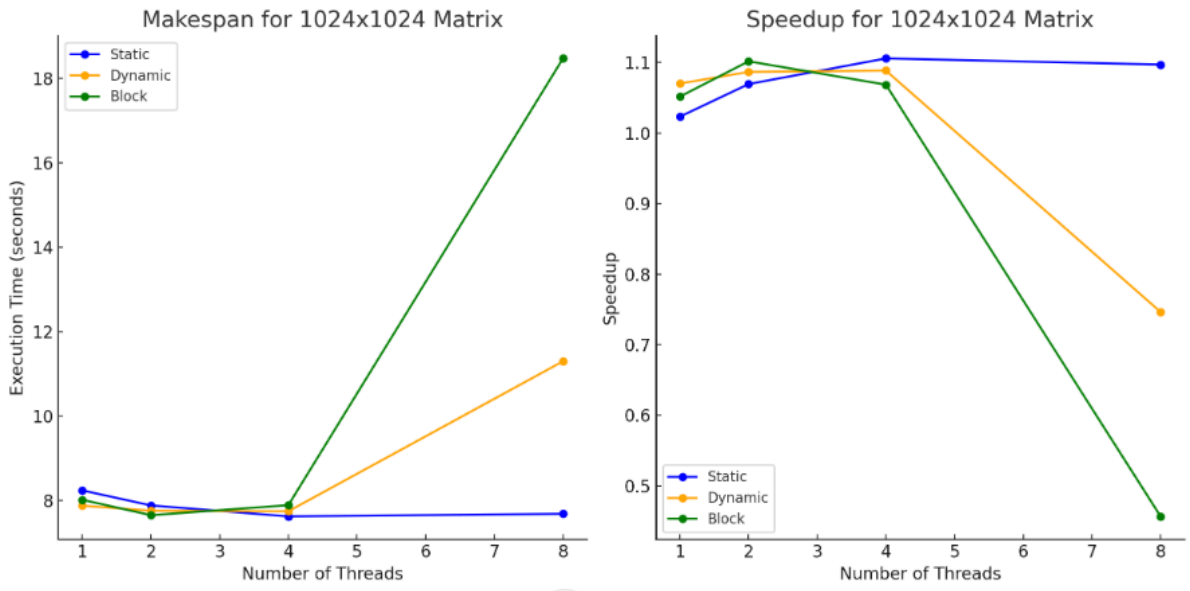
Interprétations des courbes pour chaque taille de matrice

Matrice 512x512



- **Courbe de Makespan**
    - Le temps d'exécution diminue globalement avec l'augmentation du nombre de threads, mais l'amélioration est limitée après 4 threads.
    - La stratégie **par blocs** est légèrement plus rapide que les autres pour les configurations multi-threads.
    - La stratégie **dynamique** est comparable à **par blocs** mais peut montrer une légère hausse de temps à 8 threads.
- **Courbe de Speedup**
    - Le gain en performance est optimal pour 1 à 4 threads, mais le speedup plafonne ou diminue légèrement au-delà.
    - La stratégie **statique** offre un meilleur speedup à 8 threads, tandis que **dynamique** et **par blocs** montrent des gains moindres à ce niveau.

Matrice 1024x1024

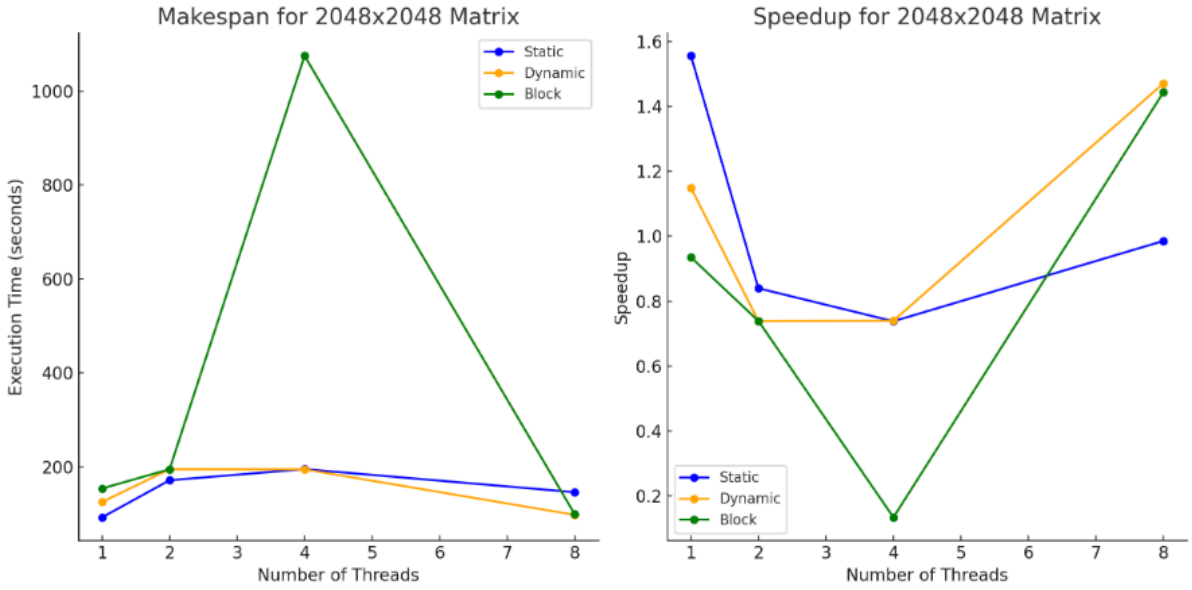


- **Courbe de Makespan**
    - Les stratégies **dynamique** et **par blocs** sont plus rapides pour 2 à 4 threads, mais **statique** devient moins performante à partir de 4 threads.

- Au-delà de 4 threads, les temps d'exécution stagnent ou augmentent légèrement.

- **Courbe de Speedup**
  - Le speedup est plus linéaire que pour la matrice 512x512 jusqu'à 4 threads, mais on observe un plafonnement au-delà.
  - Les stratégies **dynamique** et **par blocs** continuent de montrer des gains significatifs avec 4 threads, contrairement à **statique** qui se stabilise plus tôt.

Matrice 2048x2048



- **Courbe de Makespan**
  - La stratégie **statique** est nettement plus rapide jusqu'à 4 threads. **Dynamique** dépasse **statique** à 8 threads, tandis que **par blocs** devient inefficace au-delà de 2 threads.
  - On observe une dégradation marquée des performances de la stratégie **par blocs** pour un grand nombre de threads.
- **Courbe de Speedup**
  - **Statique** montre un bon gain jusqu'à 4 threads, mais son speedup diminue après.
  - **Dynamique** offre le meilleur gain à 8 threads, tandis que **par blocs** montre un impact négatif au-delà de 2 threads, indiquant une surcharge de parallélisation.

Comparaison globale de l'impact de la taille des matrices

1. **Effet sur le makespan**
    - Les matrices plus grandes augmentent considérablement le temps d'exécution global, mais l'impact de l'ajout de threads est plus prononcé pour les matrices 2048x2048.
    - Les stratégies comme **par blocs** sont moins efficaces pour les grandes matrices, probablement en raison de la surcharge de gestion des blocs.
2. **Effet sur le speedup**
    - Le speedup est plus stable pour les matrices de taille moyenne (1024x1024), tandis que pour les très grandes matrices (2048x2048), les stratégies doivent être choisies avec soin (préférer **dynamique**).
    - Les petites matrices (512x512) bénéficient moins de la parallélisation, car l'overhead devient significatif par rapport à la tâche à traiter.
3. **Comportement des stratégies**
    - **Statique** : Performante pour les matrices petites à moyennes et avec peu de threads. Elle perd en efficacité pour des matrices plus grandes ou un nombre élevé de threads.
    - **Dynamique** : Offre une bonne adaptabilité, particulièrement utile pour des matrices plus grandes et un nombre élevé de threads.
    - **Par blocs** : Performante pour des tailles de matrices petites et moyennes, mais inadaptée aux grandes tailles et à de nombreux threads.

Conclusion

- **Taille des matrices** : Plus la matrice est grande, plus l'impact de la parallélisation est prononcé, mais il faut choisir des stratégies adaptées pour éviter la surcharge.
- **Stratégies** : La stratégie dynamique est généralement la plus adaptée pour les matrices de grande taille et un grand nombre de threads.

- **Nombre de threads** : Ajouter plus de threads est bénéfique jusqu'à un certain seuil (typiquement 4 pour ces expérimentations). Au-delà, les gains en speedup diminuent ou peuvent devenir négatifs.