**TP Avancé : Optimisation de calculs parallèles avec la réduction en OpenMP**

**Objectifs du TP**

Ce TP a pour objectif de vous faire explorer des scénarios complexes d'optimisation de calculs parallèles en utilisant la **réduction en OpenMP**. Vous allez non seulement mettre en pratique les concepts de réduction, mais aussi explorer des notions plus avancées comme la gestion des dépendances, l'équilibrage de charge entre threads et la réduction des temps d'attente.

Les objectifs sont :

1. **Comprendre les concepts d'optimisation** dans des environnements de calcul parallèle avec OpenMP.
2. **Mettre en place des mécanismes d'optimisation** pour minimiser les goulots d'étranglement, notamment en gérant correctement les variables partagées et locales.
3. **Évaluer l'impact de différentes stratégies de réduction** sur la performance et l'efficacité des threads.
4. **Expérimenter avec des tailles de données et des modèles de charge de travail variables** pour observer les gains de performances.

**Description du travail demandé**

Vous allez implémenter un algorithme de calcul numérique massif, comme l'algorithme de **Monte-Carlo** ou le calcul de **Mandelbrot**, et optimisez le parallélisme en utilisant les mécanismes de réduction OpenMP. À travers plusieurs étapes, vous explorerez comment les optimisations affectent les performances.

Voici les étapes principales :

1. **Phase 1 : Implémentation d'un calcul parallèle simple avec réduction**
   * Vous commencerez par une implémentation simple de l'algorithme de Monte-Carlo pour estimer la valeur de π.
   * Utilisation de la réduction en OpenMP pour paralléliser le calcul du nombre de points à l'intérieur du cercle.
2. **Phase 2 : Optimisation des performances**
   * Vous utiliserez différentes stratégies d'optimisation : gestion des dépendances, équilibrage des charges (schedule), et limitation des conflits de cache.
   * Vous testerez différentes tailles de problème et configurations de threads pour analyser les gains de performance.
3. **Phase 3 : Mesure et analyse des performances**
   * Mesure du **speedup** et de l'**efficacité** en augmentant le nombre de threads et en comparant les versions optimisées et non optimisées.
   * Analyse des points de contention et des goulots d'étranglement dans le calcul parallèle.

**Étapes détaillées**

**Étape 1 : Implémentation de l'algorithme de Monte-Carlo pour calculer π**

L'idée de cet algorithme est de générer des points aléatoires dans un carré, puis de calculer combien de ces points tombent dans un cercle inscrit dans ce carré. Le rapport entre les points dans le cercle et les points totaux permet d'estimer la valeur de π.

* **Description** : Chaque thread générera un ensemble de points aléatoires et comptera le nombre de points qui tombent à l'intérieur du cercle. La réduction permettra de cumuler le total de tous les threads.

**Étape 2 : Optimisation des performances**

L'optimisation se concentrera sur plusieurs points clés :

1. **Équilibrage des charges** : Utilisez la clause schedule(dynamic, chunk\_size) pour ajuster la répartition des itérations entre threads. Testez différents chunk\_size pour voir l'impact sur les performances.
2. **Réduction des conflits de cache** : Les accès répétés à des variables partagées, comme dans les opérations de réduction, peuvent causer des ralentissements dus à la **falsification du cache** (cache false sharing). Une solution consiste à utiliser des variables locales par thread puis les combiner à la fin de la section parallèle.
3. **Minimisation des dépendances** : Limitez les accès aux variables partagées en donnant à chaque thread sa propre copie locale des données, puis effectuez une réduction à la fin pour minimiser les dépendances entre threads.

**Étape 3 : Mesure et analyse des performances**

1. **Mesure des performances** : Utilisez omp\_get\_wtime() pour mesurer le temps d'exécution des versions optimisées et non optimisées. Testez différentes tailles de données et différents nombres de threads.
2. **Calcul du speedup** : Le speedup est défini comme le rapport du temps d'exécution séquentiel sur le temps d'exécution parallèle.

Speedup=​​ temps d'exécution séquentiel / temps d'exécution parallèle

Calculez et comparez le speedup pour différents nombres de threads.

1. **Analyse des résultats** : Identifiez les facteurs limitant les performances (faux partage, dépendances, etc.). Discutez de l’impact du nombre de threads et des stratégies de planification (schedule) sur les performances globales.

**Exemple de questions pour guider votre analyse**

* Quel est l'impact de l'utilisation de schedule(dynamic) comparé à schedule(static) sur les performances ?
* Comment la taille des chunks affecte-t-elle la performance ? Pourquoi ?
* Qu'observez-vous en termes de performance lorsque vous augmentez le nombre de threads au-delà du nombre de cœurs physiques disponibles ?
* Quels sont les goulots d'étranglement identifiés dans votre programme et comment pouvez-vous les résoudre ?

**Résultats attendus**

* Un rapport décrivant vos observations, y compris des graphiques montrant le speedup en fonction du nombre de threads.
* Discussion des stratégies d'optimisation adoptées et leur impact sur les performances.
* Suggestions d'améliorations supplémentaires si le temps le permet.