Projet OpenMP : Simulation Parallèle de Propagation de Feu de Forêt

Introduction

Le présent projet a pour objectif de vous initier au calcul parallèle à travers le développement d'une simulation de la propagation d'un feu de forêt, en utilisant la bibliothèque OpenMP. Ce thème, visuellement évocateur et pertinent, vous permettra d'appréhender les fondamentaux du parallélisme de manière progressive et appliquée.

Objectifs Pédagogiques

Ce projet est conçu pour vous permettre de développer les compétences suivantes :

- **Compréhension des principes du parallélisme :** Acquérir une connaissance approfondie des motivations et des mécanismes du calcul parallèle.
- **Maîtrise d'OpenMP :** Utiliser efficacement les directives OpenMP (`#pragma omp for`, `#pragma omp parallel`, `#pragma omp sections`, etc.) pour paralléliser des boucles et des sections de code.
- **Gestion des dépendances de données :** Identifier et résoudre les problèmes de dépendances de données afin de prévenir les conditions de concurrence (race conditions).
- **Évaluation de la performance :** Mesurer l'accélération (speedup) et l'efficacité de votre implémentation parallèle par rapport à une version séquentielle.
- **Programmation en C/C++ :** Consolider vos compétences en programmation dans ces langages.
- **Modélisation simplifiée :** Apprendre à modéliser un phénomène physique élémentaire.

Description du Projet : Simulateur de Propagation de Feu de Forêt

Vous développerez un simulateur qui modélise une forêt sous la forme d'une grille 2D. Chaque cellule de cette grille représentera un état distinct :

- **Arbre sain**

- **Arbre en feu**
- **Cendre** (arbre consumé)
- **Vide** (absence d'arbre)

La propagation du feu s'effectuera d'une cellule en feu vers ses voisins (haut, bas, gauche, droite, et optionnellement diagonales) selon des règles de probabilité définies (ex: probabilité de propagation, influence du vent).

Étapes Fondamentales (pour une initiation progressive) :

1. **Initialisation de l'Environnement (Configuration de la Forêt) :**
    - Mettre en place une grille 2D (tableau ou `std::vector<std::vector<...>>` en C++).
    - Initialiser aléatoirement les cellules avec des arbres sains ou des espaces vides.
    - Déclencher un ou plusieurs foyers d'incendie initiaux.
2. **Déroulement Séquentiel de la Propagation (Règles de Propagation Séquentielle) :**
    - Pour chaque pas de temps de la simulation :
        - Les arbres en feu passent à l'état de cendre.
        - Les arbres sains adjacents à une cellule en feu ont une probabilité de prendre feu.
        - Les cellules de cendre conservent leur état.
    - Afficher l'état de la forêt après chaque pas de temps (par exemple, en utilisant des caractères distincts pour chaque état : `T` pour arbre, `F` pour feu, `C` pour cendre, `.` pour vide).
3. **Optimisation par Parallélisation (Implémentation OpenMP) :**
    - Identifier les boucles critiques dans votre code où la parallélisation est la plus pertinente (notamment celles qui parcourent la grille pour mettre à jour les états des cellules).
    - Appliquer la directive `#pragma omp parallel for` pour exécuter ces boucles en parallèle.
    - **Défi majeur :** La propagation du feu dépend de l'état des cellules voisines au *pas de temps précédent*. Pour gérer cette dépendance et éviter les incohérences, il est recommandé d'utiliser deux grilles : une pour l'état actuel et une pour calculer le prochain état. Une fois le calcul du prochain état terminé, la grille "prochain état" sera copiée dans la grille "actuel".
4. **Validation de la Performance (Mesure et Analyse) :**
    - Mesurer le temps d'exécution de la version séquentielle de votre simulation.

    - Mesurer le temps d'exécution de la version parallèle en variant le nombre de threads (ex: 2, 4, 8).
    - Calculer le `speedup` (accélération) et l'`efficacité` pour quantifier les gains de performance obtenus grâce à la parallélisation.

Répartition des Tâches et Collaboration via GitHub (pour 4 étudiants)

Ce projet se prête idéalement au travail d'équipe. Voici une proposition de répartition des rôles et des responsabilités, ainsi que des directives pour une collaboration efficace via GitHub :

Étudiant 1 : Le Chef de Projet / Responsable `main`

- **Rôle :** Garant de la structure globale du projet, de l'intégration du code et de la qualité de la branche principale.
- **Tâches :**
    - Créer le dépôt GitHub du projet (`kinshasa-fire-simulator`).
    - Initialiser le projet C/C++ (structure des répertoires, Makefile de base).
    - Mettre en place la structure de la grille 2D et la logique minimale de la simulation séquentielle (sans la propagation initiale).
    - **Mission principale :** Réviser et fusionner les Pull Requests des autres membres de l'équipe vers la branche `main`. Assurer la conformité du code et sa fonctionnalité avant toute fusion.
- **Branche GitHub :** `main` (pour l'initialisation et les fusions).

Étudiant 2 : Le Spécialiste de la Propagation (Propagation Core)

- **Rôle :** Développer le cœur algorithmique de la propagation du feu.
- **Tâches :**
    - Implémenter les règles détaillées de propagation du feu (transition arbre en feu -> cendre, probabilité de propagation aux voisins).
    - Gérer la logique des deux grilles (état actuel et état suivant) pour garantir une propagation correcte et sans dépendances incorrectes.
    - Mettre en place l'affichage textuel de l'état de la forêt à chaque pas de temps.
    - **Première étape de parallélisation :** Appliquer la directive `#pragma omp parallel for` sur la boucle principale de mise à jour de la grille.
- **Branche GitHub :** `feature/propagation-logic`

Étudiant 3 : L'Analyste de Performance et Configurateur (Performance & Advanced Initialization)

- **Rôle :** Mettre en œuvre les outils de mesure de performance et améliorer la flexibilité de l'initialisation de la simulation.
- **Tâches :**
  - Intégrer des fonctions de chronométrage précises pour calculer le `speedup` et l'`efficacité` (utiliser `omp_get_wtime()` d'OpenMP).
  - Développer des options d'initialisation plus flexibles pour la forêt (ex: paramétrage du pourcentage d'arbres sains, définition des points de départ du feu via des arguments).
  - **Optimisation :** Expérimenter différentes clauses `schedule` pour la directive `#pragma omp for` (ex: `static` vs `dynamic`) afin d'analyser leur impact sur la performance.
- **Branche GitHub :** `feature/performance-metrics`

Étudiant 4 : L'Explorateur de Fonctionnalités (Advanced Features)

- **Rôle :** Explorer et implémenter une ou plusieurs fonctionnalités avancées pour enrichir le réalisme ou l'interactivité de la simulation.
- **Tâches (choisir une ou deux pour commencer) :**
  - Ajouter l'**influence du vent** (modifiant la probabilité de propagation dans des directions spécifiques).
  - Intégrer des **obstacles** incombustibles (rivières, formations rocheuses) qui bloquent la propagation du feu.
  - Développer une **interface utilisateur** simple permettant de configurer les paramètres de la simulation (taille de la forêt, probabilité de propagation, etc.).
  - **Parallélisation avancée (si le temps le permet) :** Explorer l'utilisation de `#pragma omp sections` ou `#pragma omp task` pour d'autres aspects du code.
- **Branche GitHub :** `feature/advanced-features`

Protocole de Collaboration sur GitHub :

1. **L'Étudiant 1** initialise le dépôt et pousse la structure de base sur la branche `main`.
2. **Chaque étudiant** clone le dépôt et crée sa propre branche de développement à partir de `main` (`git checkout -b nom-de-votre-branche`).
3. **Chaque étudiant** développe sa partie de manière autonome sur sa branche, en effectuant des commits réguliers et avec des messages clairs.

4. Une fois une fonctionnalité implémentée et validée localement, **l'étudiant ouvre une Pull Request (PR)** depuis sa branche vers la branche `main`.
5. **L'Étudiant 1** procède à la revue de code de chaque PR, échange avec l'auteur si des modifications sont nécessaires, puis fusionne la PR dans `main` après validation.
6. **Après chaque fusion dans `main`**, il est impératif pour tous les étudiants de synchroniser leurs branches locales en tirant les dernières modifications de `main` (`git pull origin main`) avant de poursuivre le développement.

Technologies et Outils Recommandés

- **Langage de programmation :** C ou C++ (au choix).
- **Compilateur :** GCC/G++ (avec l'option de compilation `-fopenmp`).
- **Bibliothèque de parallélisation :** OpenMP.
- **Système de gestion de version :** Git et plateforme GitHub.
- **Optionnel (pour la visualisation graphique) :** SDL ou SFML pour une interface plus élaborée.

Recommandations pour un Démarrage Efficace

1. **Approche itérative :** Concentrez-vous d'abord sur la mise en œuvre fonctionnelle de la version séquentielle de votre composant avant d'introduire la parallélisation.
2. **Tests rigoureux :** Effectuez des tests fréquents après chaque modification significative, et en particulier après chaque ajout de directives OpenMP, pour garantir la correction du code et prévenir l'introduction de bugs.
3. **Débogage visuel :** L'affichage textuel de la grille est un outil de débogage simple mais puissant pour visualiser la propagation et identifier les comportements inattendus.
4. **Maîtrise des dépendances :** La gestion des dépendances de données est l'aspect le plus critique du parallélisme. Une compréhension approfondie de la manière dont les cellules interagissent et de l'impact sur la parallélisation est essentielle. La technique des deux grilles est une méthode robuste pour débuter.
5. **Quantification des gains :** La mesure et la présentation de vos résultats de performance (speedup, efficacité) sont cruciales pour démontrer l'impact de votre travail de parallélisation.