Performance Evaluation of OpenMP Programs

Course: UCS645 – Parallel & Distributed Computing

Assignment: OpenMP Parallel Programming Experiments

Introduction

This project report presents the implementation and performance analysis of three computational problems using **OpenMP** on a shared-memory multicore system. The primary objective of the assignment is to understand how parallel programming concepts such as loop parallelization, synchronization, reduction, load balancing, and performance measurement affect real-world scientific and bioinformatics applications.

The experiments focus on: 1. Molecular Dynamics force computation using Lennard–Jones potential 2. DNA sequence alignment using the Smith–Waterman algorithm 3. Heat diffusion simulation using the finite difference method

Each problem is analyzed from the perspective of **data dependencies, race conditions, scalability, and efficiency**, as required in parallel and distributed computing.

Question 1: Molecular Dynamics – Force Calculation

Problem Overview

In molecular dynamics simulations, particles interact with each other through physical forces. The Lennard–Jones (LJ) potential is commonly used to model short-range repulsion and long-range attraction between particles. Given **N particles in 3D space**, the task is to compute: - Total potential energy of the system - Forces acting on each particle

The computation involves evaluating pairwise interactions, resulting in an **O(N^2)** algorithm.

Parallelization Strategy

- The outer loop over particles is parallelized using **OpenMP** `parallel for`.
- To avoid redundant computation, only interactions with `j > i` are evaluated.
- Since each interaction updates forces on *two particles*, race conditions arise if multiple threads update the same force vector.

Handling Race Conditions

Race conditions are resolved by: - Maintaining **thread-private force arrays** for each thread - Accumulating force contributions locally - Merging all thread-local force arrays after the parallel region

This approach avoids expensive critical sections and ensures correctness.

Reduction for Total Energy

The total Lennard–Jones potential energy is accumulated using:

```
#pragma omp parallel reduction(+:total_energy)
```

This allows each thread to compute partial energy independently and combine results efficiently.

Load Balancing

The workload is uneven because particle interaction counts decrease as indices increase. To mitigate load imbalance:

```
#pragma omp for schedule(dynamic)
```

Dynamic scheduling ensures better utilization of threads by redistributing work at runtime.

Performance Measurement

Execution time is measured using `omp_get_wtime()` to evaluate scalability, speedup, and efficiency.

Conclusion (Q1)

The OpenMP-based molecular dynamics implementation demonstrates how careful handling of race conditions and dynamic scheduling can significantly improve performance while maintaining correctness. The approach scales well for moderate particle counts on shared-memory systems.

Question 2: Bioinformatics – DNA Sequence Alignment (Smith–Waterman)

Problem Overview

The Smith–Waterman algorithm performs **local sequence alignment** using dynamic programming. It constructs a scoring matrix where each cell depends on its **top, left, and top-left neighbors**.

This inherent dependency structure makes straightforward parallelization difficult.

Dependency Challenge

- Traditional row-wise or column-wise parallelization is not valid
- Each matrix cell depends on previously computed cells

Wavefront Parallelization

The solution uses **wavefront (anti-diagonal) parallelization**: - Cells on the same diagonal (`i + j = constant`) are independent - Diagonals are processed sequentially - Cells within a diagonal are computed in parallel

---

Parallel Implementation

- The outer loop iterates over diagonals
- Inner loop is parallelized using OpenMP
- A reduction operation is used to track the maximum alignment score

---

Scheduling Strategy

Static scheduling is used because: - Each diagonal has roughly similar workload - It minimizes scheduling overhead

Dynamic scheduling may be used for very large or irregular sequences.

Performance Measurement

Execution time is measured using `omp_get_wtime()`. Speedup is limited by the number of diagonals and dependency constraints.

Conclusion (Q2)

Wavefront parallelization preserves correctness while enabling parallel execution of the Smith–Waterman algorithm. Although scalability is limited by data dependencies, meaningful performance gains are achieved on multicore systems.

Question 3: Scientific Computing – Heat Diffusion Simulation

Problem Overview

The heat diffusion problem models temperature propagation on a **2D metal plate** using the finite difference method. The simulation updates temperature values over multiple time steps based on neighboring grid points.

Data Dependency Analysis

- Each grid point depends only on values from the **previous time step**
- No dependencies exist within the same time step

This makes the spatial loops ideal for parallelization.

Parallelization Strategy

- Time loop remains sequential
- Spatial loops (`i`, `j`) are parallelized using OpenMP
- **Double buffering** is used to prevent overwriting data

Reduction Operation

To compute the total heat energy of the plate, a reduction operation is applied:

```
#pragma omp parallel for reduction(+:total_heat)
```

This demonstrates correct use of OpenMP reduction for global metrics.

Scheduling Strategy

Static scheduling is chosen because: - Each grid cell requires identical computation - It provides good cache locality and minimal overhead

---

Performance Measurement

Execution time is measured using `omp_get_wtime()`. The simulation is primarily memory-bound, and scalability is limited by memory bandwidth rather than computation.

Conclusion (Q3)

The heat diffusion simulation efficiently exploits data parallelism across spatial dimensions. The absence of race conditions and use of reduction operations make it a clean and scalable OpenMP application.

Overall Conclusion

This project demonstrates the practical application of OpenMP to real-world computational problems. Each experiment highlights a different aspect of parallel programming: - **Race condition handling and load balancing** (Molecular Dynamics) - **Dependency-aware parallelization** (Smith–Waterman) - **Data-parallel stencil computation** (Heat Diffusion)

Through careful design and performance measurement, the experiments show how theoretical concepts such as speedup, efficiency, and scalability manifest in practice on shared-memory systems.

End of Report