Exercise 9

Darina Öö

25th of March 2024

1 Introduction

The objective of this exercise is to explore the parallelization of a 1-dimensional molecular dynamics simulation from Exercise 8 using OpenMP. The focus is on optimizing particle loop iterations to improve computational efficiency.

2 Code Description

The parallelized version in the file md_1d_openmp.cpp introduces OpenMP directives to enable concurrent execution across multiple threads. The core modifications involve the parallelization of loops that iterate over particles, computing their acceleration, potential energy, and kinetic energy. Critical sections ensure thread-safe updates where necessary.

```
// Main simulation loop with OpenMP parallelization
#pragma omp parallel for private(j, k, dxl, dxr, u, a)
for (int i = 0; i < local_nat; i++) {
    double u;
    accel(local_nat, i, u, a[i], box, x);
    ep[i] = u;
}
```

Listing 1: Main simulation loop with OpenMP

3 Running Instructions

To compile the OpenMP-enabled program, use the following command:

```
g++ -fopenmp md_1d_openmp.cpp -o md_1d_openmp
```

To execute the program with a specific number of threads, set the `OMP_NUM_THREADS` environment variable before running the executable. For example, to run with 4 threads:

```
export OMP_NUM_THREADS=4
./md_1d_openmp 100 0.01 1000 1.0 100
```

4 Performance Analysis

For four threads introduction of OpenMP significantly reduces the simulation's execution time. The output indicates fluctuating energy values due to the dynamic nature of the system, yet the total energy remains relatively constant, demonstrating conservation within the simulation. Notably, the simulation completes in a fraction of the time compared to the non-parallelized version, highlighting the effectiveness of parallel processing.

```
Simulation completed in 0.005108 seconds
```

This result showcases the substantial speedup achieved through parallelization, making it possible to simulate larger systems or longer time scales more feasibly.

I tested the performance impact of varying the number of threads. The following table summarizes the simulation completion times for different thread counts, which could be found in the files output_(number of threads)t.txt.

Number of Threads	Simulation Time (seconds)
1	0.00838
2	0.006492
6	0.15163
8	0.156501
12	0.671165

The results show an unexpected increase in execution time with a higher number of threads beyond a certain point. This behavior could be due to the overhead associated with managing a large number of threads or contention for shared resources. The optimal number of threads for this particular simulation on the tested hardware appears to be 2, suggesting a potential area for further investigation and optimization. Parallelizing the 1D molecular dynamics simulation with OpenMP successfully reduced the execution time with a moderate number of threads. However, the increase in execution time with too many threads highlights the importance of tuning the parallelization strategy to the specific computational workload and hardware capabilities.