HIGH PERFORMANCE COMPUTING PROJECT REPORT

Team Members :
Siva Sampreeth Josyula (S20230010110)
Sai Manideep Putchanutala (S20230010195)
Prudhvi Sai Ram Kukkala (S20230010129)

Parallelization of Lattice Boltzmann Method Solver and 3D Mean Blur Stencil

1. Problem Statement

This project addresses the parallelization of the Lattice Boltzmann Method (LBM) solver and a 3D mean blur stencil. Both are stencil-based computations with strong data-parallel characteristics, making them suitable for High Performance Computing (HPC) platforms.The objective is to analyze performance bottlenecks in the sequential version, identify parallelizable hotspots, and implement parallelization using OpenMP and CUDA.

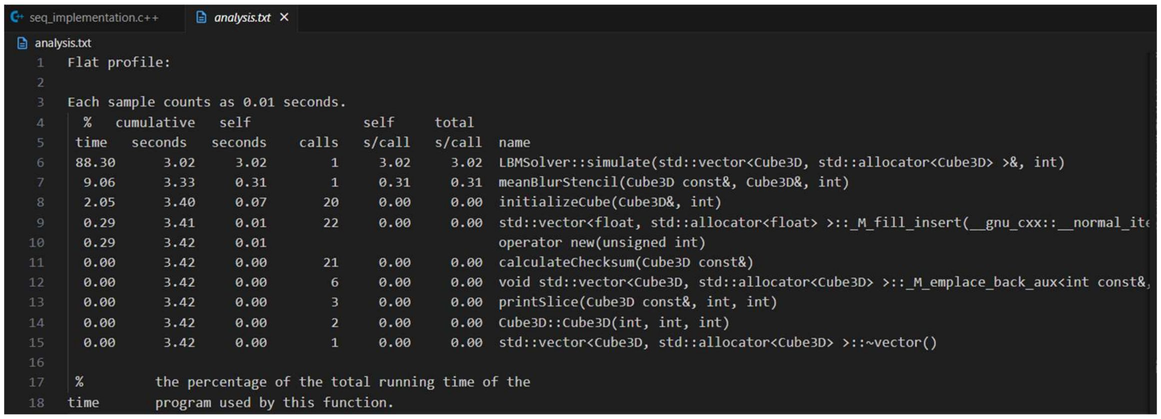
2. Sequential Implementation

A sequential C++ implementation was developed as the baseline.
It includes:
- Full LBM simulation (collision + streaming)
- Configurable 3D mean blur stencil

The sequential version works correctly but exhibits high runtime due to nested loops and cell-wise computations.

3. Performance Analysis (Sequential Profiling) : GPROF Analysis

We identified the following hotspots using gprof :



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 88.30      3.02     3.02        1     3.02     3.02  LBMSolver::simulate(std::vector<Cube3D, std::allocator<Cube3D> >&, int)
  9.06      3.33     0.31        1     0.31     0.31  meanBlurStencil(Cube3D const&, Cube3D&, int)
  2.05      3.40     0.07       20     0.00     0.00  initializeCube(Cube3D&, int)
  0.29      3.41     0.01       22     0.00     0.00  std::vector<float, std::allocator<float> >::_M_fill_insert(__gnu_cxx::__normal_ite
  0.29      3.42     0.01                             operator new(unsigned int)
  0.00      3.42     0.00       21     0.00     0.00  calculateChecksum(Cube3D const&)
  0.00      3.42     0.00        6     0.00     0.00  void std::vector<Cube3D, std::allocator<Cube3D> >::_M_emplace_back_aux<int const&,
  0.00      3.42     0.00        3     0.00     0.00  printSlice(Cube3D const&, int, int)
  0.00      3.42     0.00        2     0.00     0.00  Cube3D::Cube3D(int, int, int)
  0.00      3.42     0.00        1     0.00     0.00  std::vector<Cube3D, std::allocator<Cube3D> >::~vector()

 %         the percentage of the total running time of the
time       program used by this function.
```

1)LBMSolver::simulate() function takes 88.3% runtime

- Nested loops updating 3D grid cells. These loops consume lot of time.
- Each cell operation is independent .So there is scope for parallelization.

2)meanBlurStencil() function takes 9.06% runtime

- It includes Stencil Operation involving averaging of neighbourhood cells.
- Independent per-cell stencil operations are involved.So they are Parallelizable.

Other functions like initializing ,I/O, utility functions are not worth parallelizing.

Conclusion: Both LBM simulation and stencil operations are parallelizable and offer high performance improvement potential.

4. Chosen Parallelization Platforms

Based on the profiling using GPROF :

OpenMP (CPU shared-memory parallelization)
- Ideal for parallelizing nested loops across cores
- Simple integration with existing C++ code
- Suitable for quick speedups on multi-core CPUs

CUDA (GPU acceleration)
- Best suited for massive parallelism in grid-based computations
- Thread-per-cell execution model fits stencil patterns
- Optimized memory hierarchy enables significant speedup

Justification: Combining OpenMP and CUDA ensures scalability across CPU platforms, aligning with research findings on stencil optimizations in heterogeneous environments

5. Conclusion

We analyzed sequential performance, identified hotspots, and selected OpenMP and CUDA as parallelization platforms. These approaches target both CPU and GPU architectures, ensuring scalability and optimal performance for stencil computations such as LBM and 3D mean blur operations.Final evaluation will include comparative results to demonstrate achieved speedups.