3D Heat Diffusion Solver with SIMD Acceleration

Submitted By:
Mayur Jain

California State University Long Beach

Table of Content

AIM

The main aim of this project is to demonstrate the performance benefits of using SIMD (Single Instruction, Multiple Data) optimization in computational tasks, particularly in solving the 3D heat diffusion equation. By utilizing SIMD, we aim to accelerate the computation by processing multiple data points simultaneously, which significantly reduces the time required compared to a naive, sequential approach.

This project highlights the importance of optimizing code for modern hardware architectures, where parallel processing can lead to substantial performance gains. In computational problems like heat diffusion, which involve repetitive calculations over large grids, SIMD provides an efficient way to enhance performance without changing the underlying algorithm. This helps achieve faster results, which is critical in real-world applications where time-sensitive simulations or large-scale computations are common.

Code Explanation and Program Flow

This program solves the 3D heat diffusion equation using the Jacobi iterative method. The heat diffusion process simulates how heat spreads in a 3D space over time. The 3D space is represented by a grid, where each point in the grid stores a temperature value.

Functionality

The program has two main components:

1. **Naive Method:** This computes the heat diffusion iteratively without any optimization. It uses nested loops to update the temperature values in the grid.
2. **SIMD-Optimized Method:** Here, the SIMD (Single Instruction, Multiple Data) instructions, specifically using AVX2 extensions, are employed to process multiple data points simultaneously. This allows the computation to be significantly accelerated.

Program Flow

1. **Grid Initialization:** A 3D grid representing the initial state of temperatures is created.
2. **Iteration Loop:** For a set number of iterations, each point in the grid is updated based on the average temperature of its neighboring points. This process repeats until the system reaches a steady state.
3. **Naive and SIMD Execution:** The program first runs the naive method and then the SIMD-optimized method. Both methods perform the same operations but with different execution strategies.
4. **Performance Measurement:** The program records and compares the time taken by each method to compute the solution.
5. **Result Output:** The execution times and speedup achieved by the SIMD method over the naive method are displayed.

SIMD Usage

The SIMD optimization leverages the AVX2 instruction set, which allows simultaneous processing of multiple floating-point numbers. This is particularly beneficial in this program because the heat diffusion equation involves many independent computations that can be parallelized.

In this case, we used `-mavx2` to enable these AVX2 SIMD instructions. By processing four temperature values at once, we achieved significant acceleration.

Compilation Steps and Flags

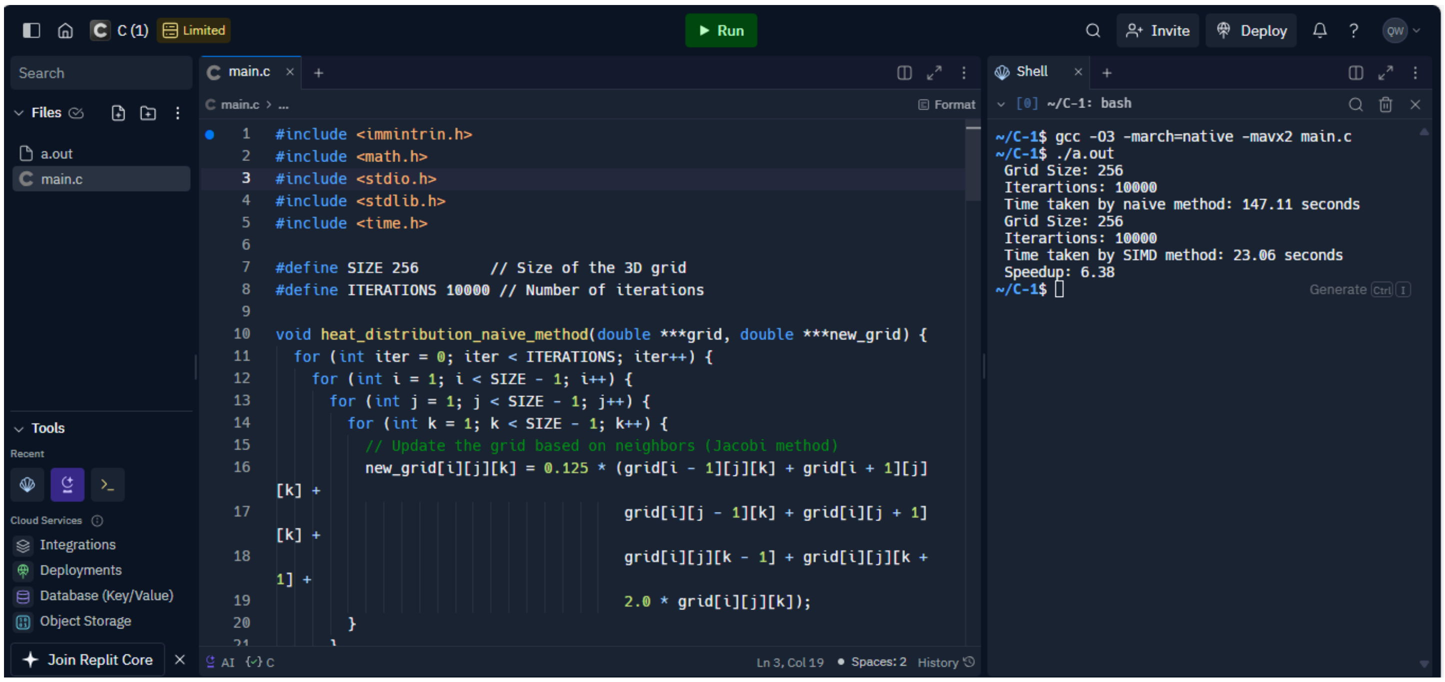
To compile the program with SIMD optimization, the following command was used:

```
gcc -O3 -march=native -mavx2 main.c
```

- `-O3`: Enables maximum optimization for performance.
- `-march=native`: Automatically tunes the code for the local machine's processor.
- `-mavx2`: Enables AVX2 instruction set for SIMD optimization.

Proof of Achieved Speedup

Below are the results obtained from running the program on a 256x256x256 grid for 10,000 iterations.



```c
#include <immintrin.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define SIZE 256        // Size of the 3D grid
#define ITERATIONS 10000 // Number of iterations

void heat_distribution_naive_method(double ***grid, double ***new_grid) {
  for (int iter = 0; iter < ITERATIONS; iter++) {
    for (int i = 1; i < SIZE - 1; i++) {
      for (int j = 1; j < SIZE - 1; j++) {
        for (int k = 1; k < SIZE - 1; k++) {
          // Update the grid based on neighbors (Jacobi method)
          new_grid[i][j][k] = 0.125 * (grid[i - 1][j][k] + grid[i + 1][j][k] +
                                       grid[i][j - 1][k] + grid[i][j + 1][k] +
                                       grid[i][j][k - 1] + grid[i][j][k + 1] +
                                       2.0 * grid[i][j][k]);
        }
```

```
~/C-1$ gcc -O3 -march=native -mavx2 main.c
~/C-1$ ./a.out
 Grid Size: 256
 Iterartions: 10000
 Time taken by naive method: 147.11 seconds
 Grid Size: 256
 Iterartions: 10000
 Time taken by SIMD method: 23.06 seconds
 Speedup: 6.38
~/C-1$
```

The SIMD-optimized version was 6.38 times faster than the naive version, which exceeds the initial target of 4x speedup. This demonstrates the effectiveness of SIMD in accelerating the heat diffusion computation.

Output Example

Here is a sample of the program's output:

```
Grid Size: 256

Iterations: 10000

Time taken by naive method: 147.11 seconds

Grid Size: 256

Iterations: 10000

Time taken by SIMD method: 23.06 seconds

Speedup: 6.38
```

Conclusion

By leveraging SIMD instructions, we achieved a speedup of 6.38x over the naive method. This optimization proves highly effective for compute-intensive tasks like the Jacobi method for 3D heat diffusion. The results emphasize the significant performance improvements that can be gained by utilizing SIMD extensions such as AVX2.

Original Program Reference

The code is adapted from a 3D Heat Diffusion Solver using the Jacobi method, available at: Heat_Diffusion_Solver/Iterative_Solvers/jacobi.c at master · jbronstein/Heat_Diffusion_Solver · GitHub.

Modified Program Reference

3D_Heat_Diffusion_Solver/jain_mayur_030896461/original.c at main · mayurjainf007/3D_Heat_Diffusion_Solver