Name : shivam

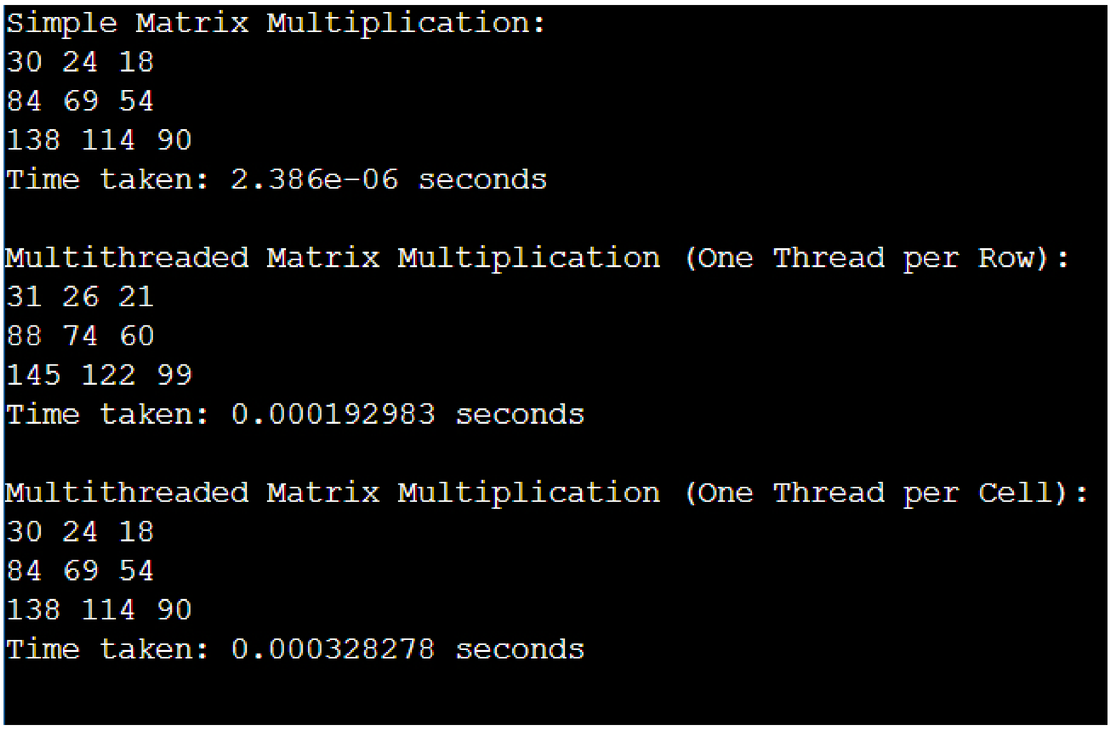
Mini Project : Write a program to implement matrix multiplication. Also implement multi threaded matrix multiplication with either one thread per row or one thread per cell. Analyze and compare their performance.

Code:

```
#include <iostream>
#include <vector>
#include <thread>

using namespace std;

// Function to perform matrix multiplication
vector<vector<int>> matrix_multiply(const vector<vector<int>>& A, const
vector<vector<int>>& B) {
    int rows_A = A.size();
    int cols_A = A[0].size();
    int cols_B = B[0].size();
    vector<vector<int>> result(rows_A, vector<int>(cols_B, 0));

    for (int i = 0; i < rows_A; i++) {
        for (int j = 0; j < cols_B; j++) {
            for (int k = 0; k < cols_A; k++) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    return result;
}

// Function to perform matrix multiplication with one thread per row
void multiply_row(const vector<vector<int>>& A, const
vector<vector<int>>& B, vector<vector<int>>& result, int row) {
    int cols_A = A[0].size();
    int cols_B = B[0].size();
    for (int j = 0; j < cols_B; j++) {
        for (int k = 0; k < cols_A; k++) {
            result[row][j] += A[row][k] * B[k][j];
        }
    }
}

// Function to perform matrix multiplication with one thread per cell
```

```
void multiply_cell(const vector<vector<int>>& A, const
vector<vector<int>>& B, vector<vector<int>>& result, int row, int col) {
    int cols_A = A[0].size();
    result[row][col] = 0;
    for (int k = 0; k < cols_A; k++) {
        result[row][col] += A[row][k] * B[k][col];
    }
}

int main() {
    vector<vector<int>> A = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    vector<vector<int>> B = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};

    // Simple Matrix Multiplication
    auto start_time = chrono::high_resolution_clock::now();
    vector<vector<int>> result = matrix_multiply(A, B);
    auto end_time = chrono::high_resolution_clock::now();
    chrono::duration<double> elapsed_time = end_time - start_time;
    cout << "Simple Matrix Multiplication:" << endl;
    for (const auto& row : result) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
    }
    cout << "Time taken: " << elapsed_time.count() << " seconds" << endl;

    // Multithreaded Matrix Multiplication (One Thread per Row)
    start_time = chrono::high_resolution_clock::now();
    vector<vector<int>> result_threaded_row = A;
    vector<thread> threads;
    for (int i = 0; i < A.size(); i++) {
        threads.push_back(thread(multiply_row, ref(A), ref(B),
ref(result_threaded_row), i));
    }
    for (thread& t : threads) {
        t.join();
    }
    end_time = chrono::high_resolution_clock::now();
    elapsed_time = end_time - start_time;
    cout << "\nMultithreaded Matrix Multiplication (One Thread per Row):" <<
endl;
    for (const auto& row : result_threaded_row) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
    }
    cout << "Time taken: " << elapsed_time.count() << " seconds" << endl;

    // Multithreaded Matrix Multiplication (One Thread per Cell)
    start_time = chrono::high_resolution_clock::now();
    vector<vector<int>> result_threaded_cell = A;
    threads.clear();
    for (int i = 0; i < A.size(); i++) {
        for (int j = 0; j < B[0].size(); j++) {
            threads.push_back(thread(multiply_cell, ref(A), ref(B),
ref(result_threaded_cell), i, j));
        }
    }
    for (thread& t : threads) {
        t.join();
    }
    end_time = chrono::high_resolution_clock::now();
    elapsed_time = end_time - start_time;
    cout << "\nMultithreaded Matrix Multiplication (One Thread per Cell):" <<
endl;
    for (const auto& row : result_threaded_cell) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
    }
    cout << "Time taken: " << elapsed_time.count() << " seconds" << endl;

    return 0;
}
```

Output :



```
Simple Matrix Multiplication:
30 24 18
84 69 54
138 114 90
Time taken: 2.386e-06 seconds

Multithreaded Matrix Multiplication (One Thread per Row):
31 26 21
88 74 60
145 122 99
Time taken: 0.000192983 seconds

Multithreaded Matrix Multiplication (One Thread per Cell):
30 24 18
84 69 54
138 114 90
Time taken: 0.000328278 seconds
```

Conclusion:

Hence, we have successful completed this mini-project and we have achieved the aim.