```
#include <mpi.h>
#include <iostream>
#include <cstdlib>
#define N 1024

using namespace std;

void matrix_mult(double* A_local, double* B, double* C_local, int rows_per_process) {
    for (int i = 0; i < rows_per_process; i++) {
        for (int j = 0; j < N; j++) {
            C_local[i * N + j] = 0;
            for (int k = 0; k < N; k++) {
                C_local[i * N + j] += A_local[i * N + k] * B[k * N + j];
            }
        }
    }
}

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int rows_per_process = N / size;

    double* A_local = new double[rows_per_process * N];
    double* B = new double[N * N];
    double* C_local = new double[rows_per_process * N];

    if (rank == 0) {
        double* A = new double[N * N];
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                A[i * N + j] = rand() % 100;
                B[i * N + j] = rand() % 100;
            }
        }

        MPI_Scatter(A, rows_per_process * N, MPI_DOUBLE, A_local, rows_per_process * N,
MPI_DOUBLE, 0, MPI_COMM_WORLD);
        delete[] A;
    } else {

        MPI_Scatter(nullptr, rows_per_process * N, MPI_DOUBLE, A_local, rows_per_process * N,
MPI_DOUBLE, 0, MPI_COMM_WORLD);
    }

    MPI_Bcast(B, N * N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    double start_time = MPI_Wtime();
    matrix_mult(A_local, B, C_local, rows_per_process);

    double end_time = MPI_Wtime();
    double elapsed_time = end_time - start_time;

    if (rank == 0) {
        double* C = new double[N * N];
        MPI_Gather(C_local, rows_per_process * N, MPI_DOUBLE, C, rows_per_process * N,
MPI_DOUBLE, 0, MPI_COMM_WORLD);
        delete[] C;
    } else {
        MPI_Gather(C_local, rows_per_process * N, MPI_DOUBLE, nullptr, rows_per_process * N,
MPI_DOUBLE, 0, MPI_COMM_WORLD);
    }

    if (rank == 0) {
        cout << "Time taken for matrix multiplication: " << elapsed_time << " seconds." << endl;
    }

    delete[] A_local;
    delete[] B;
    delete[] C_local;

    MPI_Finalize();
    return 0;
}
```