```cpp
#include <iostream>
#include <pthread.h>
#include <chrono>
#include <thread>

#define N 1024

int **a = new int *[N];
int **b = new int *[N];
int **c = new int *[N];
uint64_t x[N];

void *column_wise_add(void *arguments)
{
    int j = *((int *)arguments);
    for (int i = 0; i < N; i++)
    {
        c[i][j] = a[i][j] + b[i][j];
    }
    return NULL;
}
void column_wise()
{
    pthread_t threads[N];
    int thread_args[N];

    auto start = std::chrono::high_resolution_clock::now();
    for (int j = 0; j < N; j++)
    {
        thread_args[j] = j;
        pthread_create(&threads[j], NULL, column_wise_add, &thread_args[j]);
    }
    for (int j = 0; j < N; j++)
    {
        pthread_join(threads[j], NULL);
    }
    auto end = std::chrono::high_resolution_clock::now();

    double time_taken =
        std::chrono::duration_cast<std::chrono::microseconds>(end - start).count();
    std::cout << "row_wise: " << time_taken << " microseconds" << std::endl;
}
void *row_wise_add(void *arguments)
{
    int i = *((int *)arguments);
    for (int j = 0; j < N; j++)
    {
        c[i][j] = a[i][j] + b[i][j];
    }
    return NULL;
}
void row_wise()
{
    pthread_t threads[N];
    int thread_args[N];

    auto start = std::chrono::high_resolution_clock::now();
```

```
    for (int i = 0; i < N; i++)
    {
        thread_args[i] = i;
        pthread_create(&threads[i], NULL, row_wise_add, &thread_args[i]);
    }
    for (int i = 0; i < N; i++)
    {
        pthread_join(threads[i], NULL);
    }
    auto end = std::chrono::high_resolution_clock::now();

    double time_taken =
        std::chrono::duration_cast<std::chrono::microseconds>(end - start).count();
    std::cout << "row_wise: " << time_taken << " microseconds" << std::endl;
}
void serial_1()
{
    auto start = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    auto end = std::chrono::high_resolution_clock::now();
    double time_taken =
        std::chrono::duration_cast<std::chrono::microseconds>(end - start).count();
    std::cout << "serial: " << time_taken << " microseconds" << std::endl;
}
void question1()
{
    for (int i = 0; i < N; i++)
    {
        a[i] = new int[N];
        b[i] = new int[N];
        c[i] = new int[N];
    }

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            a[i][j] = 1 + (rand() % 1024);
            b[i][j] = 1 + (rand() % 1024);
        }
    }
    serial_1();
    row_wise();
    column_wise();
}

double Riemann_Zeta(double s, uint64_t k)
{
    double result = 0.0;

    for (uint64_t i = 1; i < k; i++)
    {
        for (uint64_t j = 1; j < k; j++)
        {
            result += (2 * (i & 1) - 1) / pow(i + j, s);
        }
```

```
    }
    return result * pow(2, s);
}
void Riemann_Zeta_parallel(double s, uint64_t k)
{
    double result = 0.0;

    for (uint64_t i = 1; i < k; i++)
    {
        for (uint64_t j = 1; j < k; j++)
        {
            result += (2 * (i & 1) - 1) / pow(i + j, s);
        }
    }
    x[k] = result * pow(2, s);
}
void serial_2()
{
    auto start = std::chrono::high_resolution_clock::now();
    for (uint64_t k = 0; k < N; k++)
    {
        x[k] = Riemann_Zeta(2, k);
    }
    auto end = std::chrono::high_resolution_clock::now();
    double time_taken =
        std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count();
    std::cout << "serial: " << time_taken << " milliseconds" << std::endl;
}
void c_threads()
{
    std::thread threads[N];
    auto start = std::chrono::high_resolution_clock::now();
    for (uint64_t k = 0; k < N; k++)
    {
        threads[k] = std::thread(Riemann_Zeta_parallel, 2, k);
    }
    for (uint64_t k = 0; k < N; k++)
    {
        threads[k].join();
    }
    auto end = std::chrono::high_resolution_clock::now();
    double time_taken =
        std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count();
    std::cout << "parallel: " << time_taken << " milliseconds" << std::endl;
}
void question2()
{
    serial_2();
    c_threads();
}
int main(int argc, const char *argv[])
{
    question1();

    question2();
    // No data dependencies nor shared variables.
    return 0;
}
```