Текст программы:

```
#include <iostream>
#include <algorithm>
#include <numeric>
#include <vector>
#include <thread>
#include <string>
using namespace std;

constexpr int dim = 1000000000;
constexpr int num_of_threads = 2;
constexpr int a = 3;
constexpr int b = 39;

void print_time(const string& type, const chrono::time_point<chrono::steady_clock>
start, const chrono::time_point<chrono::steady_clock> end)
{
   const auto elapsed_time = chrono::duration_cast<chrono::microseconds>(end -
start);
   cout << type + " time: " + std::to_string(elapsed_time.count()) + " ms\n";
}

double f(const double x) {
   return -x * x;
}

double integral(const double a, const double b, const int n) {
   const double h = (b - a) / n;
   double x = a, res = 0;
   for (int i = 0; i < n; i++) {
      res += f(x) * h;
      x += h;
   }
   return res;
}

struct thread_block {
   void operator()(const double a, const double b, const int n, double& result)
   {
      result = integral(a, b, n);
   }
};
```

```
double parallel_integral(const double a, const double b, const int n, const int
thread_num) {
    vector<double> integrals(thread_num);
    vector<thread> threads(thread_num - 1);
    const double step = (b - a) / thread_num;
    double part_a = a;
    for (int i = 1; i < thread_num; i++) {
        double part_b = part_a + step;
        threads[i - 1] = thread(thread_block(), part_a, part_b, n / thread_num,
ref(integrals[i - 1]));
        part_a = part_b;
    }
    thread_block()(part_a, b, n - (n / thread_num) * (thread_num - 1),
ref(integrals[thread_num - 1]));
    for_each(threads.begin(), threads.end(), [](thread& t) { t.join(); });
    return accumulate(integrals.begin(), integrals.end(), 0.0);
}


void random_data_initialization(vector<double>& a, vector<double>& b) {
    const auto size = b.size();
    srand(time(nullptr));
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            a[i * size + j] = rand() % 100;
        }
        b[i] = rand() % 100;
    }
}

struct matrix_block {
    void operator()(const unsigned int first_row, const unsigned int last_row,
        const vector<double>& matrix, const vector<double>& vec, vector<double>&
res) const
    {
        const auto size = res.size();
        for (auto i = first_row; i < last_row; i++)
            for (int j = 0; j < size; j++)
                res[i] += matrix[i * size + j] * vec[j];
    }
};

void parallel_multiply(vector<double>& m, vector<double>& v,
    vector<double>& res, const int thread_num) {
    vector<double> results(thread_num);
```

```
    vector<thread> threads(thread_num - 1);
    const auto portion = res.size() / thread_num;
    for (int i = 1; i < thread_num; i++) {
        threads[i - 1] = thread(matrix_block(), (i - 1) * portion, i * portion, ref(m),
            ref(v), ref(res));
    }
    matrix_block()(portion * (thread_num - 1), res.size(), m, v, res);
    for_each(threads.begin(), threads.end(), [](thread& t) { t.join(); });
}

void main() {
    auto start = chrono::steady_clock::now();
    cout << "Integral: " << integral(a, b, dim) << "\n";
    auto end = chrono::steady_clock::now();
    print_time("Sequence integral: ", start, end);

    start = chrono::steady_clock::now();
    cout << "Parallel integral: " << parallel_integral(a, b, dim, 2) << "\n";
    end = chrono::steady_clock::now();
    print_time("Parallel integral 2: ", start, end);

    start = chrono::steady_clock::now();
    cout << "Parallel integral: " << parallel_integral(a, b, dim, 4) << "\n";
    end = chrono::steady_clock::now();
    print_time("Parallel integral 4: ", start, end);

    vector<double> matrix(dim * dim);
    vector<double> v(dim);
    vector<double> result(dim);
    srand(time(nullptr));
    for (int i = 0; i < dim; i++) {
        for (int j = 0; j < dim; j++) {
            matrix[i * dim + j] = rand() % 100;
        }
        v[i] = rand() % 100;
    }

    auto start_ = chrono::steady_clock::now();
    parallel_multiply(matrix, v, result, 1);
    auto end_ = chrono::steady_clock::now();
    print_time("Sequence multiply: ", start, end);

    start_ = chrono::steady_clock::now();
    parallel_multiply(matrix, v, result, 2);
    end_ = chrono::steady_clock::now();
```

```
    print_time("Parallel multiply 2: ", start, end);

    start_ = chrono::steady_clock::now();
    parallel_multiply(matrix, v, result, 4);
    end_ = chrono::steady_clock::now();
    print_time("Parallel multiply 4: ", start, end);
}
```

Результаты вычислений:

		2 потока			4 потока		
Размерность задачи	Время выполнения п	Время выполнени	Ускорение	Эффективность	Время выполнени	Ускорение	Эффективность
10^5	0,003956	0,003434	1,15	0,576004659289458	0,002063	1,91759573436743	0,479398933591857
10^6	0,040325	0,02193	1,84	0,919402644778842	0,015967	2,52552138786247	0,631380346965617
10^7	4,025454	1,933213	2,08	1,04113049105298	1,013241	3,9728494997735	0,993212374943375
10^8	4,025554	1,9931231	2,02	1,00986085606052	1,3004214	3,095576556953	0,77389413923825