Никончик Даниил 3 курс 12 группа
Последовательная программа

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>
#include <chrono>

using namespace std;

int main() {
    int rows = 1000, cols = 100;
    vector<vector<int>> vec(rows, vector<int>(cols));

    mt19937 rng{ random_device{}() };
    uniform_int_distribution<int> dist{ 1, 100000 };
    for (auto i = 0; i < cols; ++i)
        for (auto j = 0; j < rows; ++j)
            vec[i][j] = dist(rng);

    auto start = chrono::high_resolution_clock::now();
    int Max = -1;
    for (auto i = 0; i < cols; ++i) {
        int Min = INFINITY;

        for (auto j = 0; j < rows; ++j)
            Min = min(Min, vec[i][j]);

        Max = max(Min, Max);
    }

    auto end = chrono::high_resolution_clock::now();
    auto res = chrono::duration_cast<chrono::microseconds>(end -
start).count();
    cout << "maximum = " << Max << " time = " << res << "ms.\n";
}
```

Параллельный вариант

```
#include <iostream>
#include <omp.h>
#include <random>
#include <climits>
#include <chrono>
#include <algorithm>

using namespace std;

int parallel_nested(const vector<vector<int>>& matrix) {
    int max_val = INT_MIN;

    omp_set_nested(true);

#pragma omp parallel for reduction(max: max_val)
    for (const auto& row : matrix) {
        int min_in_row = INT_MAX;
```

```
#pragma omp parallel for reduction(min: min_in_row)
        for (int j = 0; j < row.size(); j++)
            min_in_row = min(min_in_row, row[j]);

        max_val = max(max_val, min_in_row);
    }
    return max_val;
}

int max_min_matrix(const vector<vector<int>>& matrix) {
    int max_val = INT_MIN;

#pragma omp parallel for reduction(max: max_val)
    for (const auto& row : matrix) {
        int min_in_row = *min_element(row.begin(), row.end());
        max_val = max(min_in_row, max_val);
    }

    return max_val;
}

int main() {
    int size = 100;
    vector<vector<int>> vec(size, vector<int>(size));

    mt19937 rng{ random_device{}() };
    uniform_int_distribution<int> dist{ 1, 100000 };
    for (auto i = 0; i < size; ++i)
        for (auto j = 0; j < size; ++j)
            vec[i][j] = dist(rng);

    auto start = std::chrono::high_resolution_clock::now();
    max_min_matrix(vec);
    auto end = std::chrono::high_resolution_clock::now();
    auto res = (end - start).count();
    cout << res << " ms\n";

    start = std::chrono::high_resolution_clock::now();
    parallel_nested(vec);
    end = std::chrono::high_resolution_clock::now();
    res = (end - start).count();

    cout << res << " ms\n";
    return 0;
}
```

Результаты

Размерность	Линейная	Параллельная	Ускорение
1000*100	1002	8213	0.12367318
1000*1000	21387	9732	2.26883355
10000*10000	192123	36761	5.18614742