Доскоч Роман 3 курс 13 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;

}

Резульатыт

