Петров Андрей, 13 группа 3 курс

Изображение выглядит как текст

Автоматически созданное описание

1. Создадим последовательную программу

**int** vector\_max(vector<**int**> &vec)  
{  
 **return** \*max\_element(vec.begin(), vec.end());  
}  
  
**int** matrix\_max\_seq(vector<vector<**int**>> &matrix) {  
 **int** max\_value = 0;  
 **int** tmp\_value = 0;  
 **for**(**auto** &row : matrix) {  
 tmp\_value = vector\_max(row);  
 }  
 **if**(tmp\_value > max\_value) max\_value = tmp\_value;  
 **return** max\_value;  
}

**int** main(**int** argc, **char** \*\*argv)  
{  
 LARGE\_INTEGER liFrequency, liStartTime, liFinishTime;  
 **double** dElapsedTime;  
 QueryPerformanceFrequency(&liFrequency);  
  
 vector<vector<**int**>> matrix(NUMBER\_ROWS, vector<**int**>(NUMBER\_COLUMNS));  
 fill\_matrix(matrix, NUMBER\_ROWS, NUMBER\_COLUMNS);  
  
 QueryPerformanceCounter(&liStartTime);**int** total\_max = matrix\_max\_seq(matrix);  
 QueryPerformanceCounter(&liFinishTime);  
  
 dElapsedTime = 1000.0 \* (liFinishTime.QuadPart - liStartTime.QuadPart) / liFrequency.QuadPart;  
 printf(**"Time: %f ms\n"**, dElapsedTime);  
 printf(**"Total max: %d\n"**, total\_max);  
 **return** 0;  
}

2. Создадим параллельную программу

**int** vector\_max(vector<**int**> &vec)  
{  
 **return** \*max\_element(vec.begin(), vec.end());  
}  
  
**int** matrix\_row\_max(vector<vector<**int**>> &matrix, **int** thread\_index, **int** thread\_count)  
{  
 **int** n = matrix.size();  
 **int** r = n % thread\_count;  
 **int** start = **min**(thread\_index, r) + (n / thread\_count) \* thread\_index;  
 **int** end = **min**(thread\_index + 1, r) + (n / thread\_count) \* (thread\_index + 1);  
  
 vector<**int**> maximums;  
 **for** (thread\_index = start; thread\_index < end; ++thread\_index) {  
 maximums.emplace\_back(vector\_max(matrix[thread\_index]));  
 }  
 **return** vector\_max(maximums);  
}

**int** main(**int** argc, **char** \*\*argv)  
{  
 LARGE\_INTEGER liFrequency, liStartTime, liFinishTime;  
 **double** dElapsedTime;  
 QueryPerformanceFrequency(&liFrequency);  
  
 vector<vector<**int**>> matrix(NUMBER\_ROWS, vector<**int**>(NUMBER\_COLUMNS));  
 fill\_matrix(matrix, NUMBER\_ROWS, NUMBER\_COLUMNS);  
  
 QueryPerformanceCounter(&liStartTime);  
 vector<**int**> maximums(THREAD\_COUNT);  
 vector<future<**int**>> futures;  
 vector<thread> threads;  
  
 **for** (**auto** i = 0; i < THREAD\_COUNT; ++i) {  
 **auto** task = packaged\_task<**int**(vector<vector<**int**>> &, **int**, **int**)>(matrix\_row\_max);  
 futures.emplace\_back(task.get\_future());  
 threads.emplace\_back(move(task), ref(matrix), i, THREAD\_COUNT);  
 }  
  
 **for** (**auto** i = 0; i < THREAD\_COUNT; ++i) {  
 maximums[i] = futures[i].get();  
 threads[i].join();  
 }  
 **int** total\_max = vector\_max(maximums);  
 QueryPerformanceCounter(&liFinishTime);  
  
 dElapsedTime = 1000.0 \* (liFinishTime.QuadPart - liStartTime.QuadPart) / liFrequency.QuadPart;  
 printf(**"Time: %f ms\n"**, dElapsedTime);  
 printf(**"Total max: %d\n"**, total\_max);  
 **return** 0;  
}

Произведем вычислительные эксперименты и составим таблицу

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность матрицы | Последовательная программа | Параллельная программа | Ускорение |
| 100 x 100 | 0.140100 | 1.616700 | 11.5396146 |
| 1 000 x 1 000 | 10.062500 | 3.916900 | 0.389257 |
| 10 000 x 10 000 | 715.511200 | 275.881900 | 0.38557314 |