Петров Андрей, 13 группа 3 курс

**Изображение выглядит как текст

Автоматически созданное описание**

1. Создадим последовательную программу

**int** find\_row\_max(vector<**int**> &matrix) {  
 **int** max\_value = 0; **for** (**int** i = 0; i < matrix.size(); i++) {  
 **if** (matrix[i] > max\_value) max\_value = matrix[i];  
 }  
 **return** max\_value;  
}  
  
**float** find\_min\_value(vector<**int**> &matrix) {  
 **int** min\_value = **INT\_MAX**; **for** (**int** i = 0; i < matrix.size(); i++) {  
 **if** (matrix[i] < min\_value) min\_value = matrix[i];  
 }  
 **return** min\_value;  
}  
  
**int** find\_total\_max(vector<vector<**int**>> &matrix, **int** row) {  
 vector<**int**> min\_elements(row);  
  
 **for** (**int** i = 0; i < row; i++)   
 min\_elements[i] = find\_min\_value(matrix[i]);  
 }  
 **return** find\_row\_max(min\_elements);  
}

2. Создадим параллельную программу

**int** find\_row\_max(vector<**int**> &matrix) {  
 **int** max\_value = 0;  
  
#pragma omp parallel **for  
 for** (**int** i = 0; i < matrix.size(); i++) {  
 **if** (matrix[i] > max\_value) {  
#pragma omp critical  
 max\_value = matrix[i];  
 }  
 }  
 **return** max\_value;  
}  
  
**float** find\_min\_value(vector<**int**> &matrix) {  
 **int** min\_value = **INT\_MAX**;  
  
#pragma omp parallel **for  
 for** (**int** i = 0; i < matrix.size(); i++) {  
 **if** (matrix[i] < min\_value) {  
#pragma omp critical  
 min\_value = matrix[i];  
 }  
 }  
 **return** min\_value;  
}  
  
**int** find\_total\_max(vector<vector<**int**>> &matrix, **int** row) {  
 vector<**int**> min\_elements(row);  
  
 **for** (**int** i = 0; i < row; i++) {  
 min\_elements[i] = find\_min\_value(matrix[i]);  
 }  
 **return** find\_row\_max(min\_elements);  
}

3. Замерим время

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

Изображение выглядит как текст

Автоматически созданное описание

1. Создадим версию программы с вложенным распараллеливанием

**int** find\_total\_max\_nesting(vector<vector<**int**>> &matrix, **int** row, **int** cols) {  
 **int** max\_value = **INT\_MIN**;  
 omp\_set\_nested(**true**);  
  
#pragma omp parallel **for** reduction(max:max\_value)  
 **for** (**int** i = 0; i < row; i++) {  
 **int** min\_value = **INT\_MAX**;  
  
#pragma omp parallel **for** reduction(min:min\_value)  
 **for** (**int** j = 0; j < cols; j++) {  
 min\_value = **min**(min\_value, matrix[i][j]);  
 }  
 max\_value = **max**(max\_value, min\_value);  
 }  
 **return** max\_value;  
}

Произведем вычислительные эксперименты и составим таблицу

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размерность матрицы | Последовательная программа | Параллельная программа | Параллельная программа с вложенностью | Ускорение | |
| 1 | 2 |
| 100 x 100 | 0.230100 | 0.105100 | 0.176000 | 0.4568 | 0.76488 |
| 1 000 x  1 000 | 10.390000 | 8.761400 | 14.170500 | 0.84325 | 1.36385 |
| 10 000 x  10 000 | 796.968400 | 775.129100 | 1086.539200 | 0.97259 | 1.36334 |