

Лабораторна робота #5

Виконав: Кузьменко Владислав(КН-22)

Тема: Технологія OpenMP

Мета: Познайти з базовими директивами OpenMP та навчитись їх застосовувати для розпаралелювання послідовних програм

Теоретичні відомості.

OpenMP дозволяє легко і швидко створювати багатопоточні програми на алгоритмічних мовах Fortran і C/C++. При цьому директиви OpenMP аналогічні директивам препроцесора для мови C/C++ і є аналогом коментарів у алгоритмічній мові Fortran. Це дозволяє в будь-який момент розробки паралельної реалізації програмного продукту при необхідності повернутися до послідовного варіанту програми. OpenMP – це стандартна модель для паралельного програмування в середовищі зі спільною пам'яттю. У даній моделі усі процеси спільно використовують загальний адресний простір, до якого вони асинхронно звертаються із запитами на читання і запис. У таких моделях для управління доступом до загальної пам'яті використовуються різні механізми синхронізації типу семафорів та блокування процесів. Перевага цієї моделі з точки зору програмування полягає в тому, що поняття монопольного володіння даними відсутнє, отже, не потрібно явно задавати обмін даними між потоками, що їх задають, та потоками, що їх використовують. Ця модель, з одного боку, спрощує розробку програми, але, з іншого боку, ускладнює розуміння і управління локальністю даних, написання детермінованих програм.

Git

C++: https://github.com/x1nett/parproglab5_cpp.git

Лістинг коду

```
ConsoleApplication1 (Global Scope)
1  #include <iostream>
2  #include <omp.h>
3  #include <climits>
4
5  using namespace std;
6
7  const int ROWS = 10000;
8  const int COLS = 1000;
9
10 int matrix[ROWS][COLS];
11
12 void init_matrix() {
13     for (int i = 0; i < ROWS; ++i)
14         for (int j = 0; j < COLS; ++j)
15             matrix[i][j] = 1 + (i + j) % 100;
16
17     for (int j = 0; j < COLS; ++j)
18         matrix[ROWS / 2][j] = -100;
19 }
20
21 long long total_sum(int num_threads) {
22     long long sum = 0;
23     double t1 = omp_get_wtime();
24
25     #pragma omp parallel for reduction(+:sum) num_threads(num_threads)
26     for (int i = 0; i < ROWS; ++i)
27         for (int j = 0; j < COLS; ++j)
28             sum += matrix[i][j];
29
30     double t2 = omp_get_wtime();
31     cout << "Total sum computed in " << t2 - t1 << " seconds with " << num_threads << " threads.\n";
32
33     return sum;
34 }
35
36 void row_with_min_sum(int num_threads, int& min_row_index, long long& min_row_sum) {
37     min_row_sum = LLONG_MAX;
38     min_row_index = -1;
39
40     double t1 = omp_get_wtime();
41
42     #pragma omp parallel num_threads(num_threads)
43     {
44         int local_min_index = -1;
45         long long local_min_sum = LLONG_MAX;
46
47         #pragma omp for
48         for (int i = 0; i < ROWS; ++i) {
49             long long row_sum = 0;
50             for (int j = 0; j < COLS; ++j)
51                 row_sum += matrix[i][j];
52
53             if (row_sum < local_min_sum) {
54                 local_min_sum = row_sum;
55                 local_min_index = i;
56             }
57         }
58
59         #pragma omp critical
60         {
61             if (local_min_sum < min_row_sum) {
62                 min_row_sum = local_min_sum;
63                 min_row_index = local_min_index;
64             }
65         }
66     }
67
68     double t2 = omp_get_wtime();
69     cout << "Row with min sum computed in " << t2 - t1 << " seconds with " << num_threads << " threads.\n";
70 }
```

```

71
72  ✓ int main() {
73      init_matrix();
74      omp_set_nested(1);
75
76      int num_threads = 8;
77      long long total;
78      int min_index;
79      long long min_value;
80
81      double t_start = omp_get_wtime();
82
83      #pragma omp parallel sections
84      {
85          #pragma omp section
86          {
87              total = total_sum(num_threads);
88          }
89
90          #pragma omp section
91          {
92              row_with_min_sum(num_threads, min_index, min_value);
93          }
94      }
95
96      double t_end = omp_get_wtime();
97
98      cout << "=== Results ===\n";
99      cout << "Total matrix sum = " << total << endl;
100     cout << "Min row index = " << min_index << ", sum = " << min_value << endl;
101     cout << "Overall execution time: " << t_end - t_start << " seconds\n";
102
103     return 0;
104 }

```

результат виконання програми:

```

Microsoft Visual Studio Debug
Total sum computed in 0.0142362 seconds with 8 threads.
Row with min sum computed in 0.0141723 seconds with 8 threads.
=== Results ===
Total matrix sum = 504849500
Min row index = 5000, sum = -100000
Overall execution time: 0.0398446 seconds

C:\Users\users_rl4r6zj\source\repos\ConsoleApplication1\x64\Debug\ConsoleApplication1.exe (process 9152) exited with code 0 (0x0).
Press any key to close this window . . .|

```