Лабораторная работа № 2 OpenMP tasks	ФИО	Титов А.К.
	Группа	ИВТ 460
	Предмет	Параллельное программирование
	Дата отчета	
	Оценка	
	Подпись преподавателя	

Цель работы

OpenMP 3.0. Изучение механизма заданий (tasks) в OpenMP.

Результаты выполнения

1a) Изучить простейшую программу, использующую механизм заданий

Вывод программы

```
Single - 1
2
2
2
2
FinishedFinished2
2
Finished2
Finished2
2
Finished21
2
2
2
Finished2
Finished2
Finished2
2
Finished2
Finished02
```

Выводы

В случае программы 1а, к переменной p может получить доступ каждый поток. Следовательно, каждый поток может инициировать ситуацию прекращения цикла и вывода слова Finish

1b) Изменить программу таким образом, чтобы изменение переменной p осуществлялось одним потоком

Вывод программы

```
Task 1b
Single - 1
1
1
Finished1
```

Выводы

В случае программы 1б, переменную p может изменить лишь один поток (из-за single).
Слово Finish будет выведено лишь 1 раз.

2) Обходы дерева

Вывод программы

```
2а) Последовательная обработка
Время последовательного варианта 34.4255
Число вершин 221

2а) Последовательный и параллельный варианты
Время последовательного варианта 34.3307
Время параллельного варианта 37.253

2в) Параллельный вариант без 'pragma omp single'
Время последовательного варианта 34.1306
Время параллельного варианта 150.608

2г) Восходящий и нисходящий варианты
Нисходящая обработка
Восходящая обработка
Время нисходящей параллельной обработки 37.5572
Время восходящей параллельной обработки 36.8993
```

3) Поиск элемента

Вывод программы

```
3а) Поиск элементов abs(N - 120) < 30
Последовательный вариант 1.6e-05
Число элементов (abs(N-par) <= prec) = 61Время выполнения параллельного
варианта0.001491
Число вершин 221
Число элементов (abs(N-par) <= prec) = 61

3б) [Atomic] Поиск элементов abs(N - 120) < 30
Время посоледовательного варианта 1.6e-05
Число вершин (abs(N-par) <= prec) = 61
Время параллельного варианта (atomic) 0.000713
Число вершин 221
Число вершин (abs(N-par) <= prec) = 61

3гe) Последовательный и параллельный варианты поиска до 1 встретившейся вершины
Последовательный вариант 7.9e-05
Число вершин 124
Первая найденная 90
Параллельный вариант 0.015824
Число вершин 124
Первая найденная 90
```

Приложение А. Код программы

1.h

```
#pragma once
#include <iostream>
#include <cstdlib>
#include "omp.h"
using namespace std;

int task_1a()
{
    int p = 1;
#pragma omp parallel shared(p)
    {
#pragma omp single
        {
            cout << "Single - "<< omp_get_thread_num() << endl;
            while (p)
            {
#pragma omp task
                {
                    int n = rand() % 100;
                    if (n < 50)
                    {
                        p = 0;
                        cout << "Finished" << omp_get_thread_num()
                        << endl;
                    }
                    else
                    {
                        cout << omp_get_thread_num() << endl;
                    }
                }
            }
        }
    }
}

void task_1b(){
    int p = 1;
#pragma omp parallel shared(p)
    {
#pragma omp single
        {
            cout << "Single - "<< omp_get_thread_num() << endl;
            while (p)
            {
                int n = rand() % 100;
                if (n < 50)
                {
                    p = 0;
                    cout << "Finished" << omp_get_thread_num() << endl;
                }
                else
                {
                    cout << omp_get_thread_num() << endl;
                }
            }
        }
    }
}
```

2.h

```
1 //
2 // Created by aleksey on 03.11.16.
3 //
4
5 #ifndef TASKSOPENMP_2A_H
6 #define TASKSOPENMP_2A_H
7
8 #include "Tree.h"
9 #include <iostream>
10 #include <omp.h>
11
12 using namespace std;
13
14 // 2a
15 void node_process(struct node *tree)
16 {
17     // Подсчитываем число узлов
18     num_nodes++;
19     // Связываем с каждым узлом какую-то работу
20     // Работа имеет разную вычислительную сложность для различных вершин
21     work(tree->num);
22     // Выводим номер узла, который обработали
23     //cout << tree->num << endl;
24     if (tree->left)
25         node_process(tree->left);
26     if (tree->right)
27         node_process(tree->right);
28     return;
29 }
30
31 void task_2a(node * tree){
32     cout << "2a) Последовательная обработка" << endl;
33     clock_t start, finish; // переменные для измерения времени
34     double time;
35     start = clock();
36     node_process(tree);
37     finish = clock();
38     time = (double) (finish - start) / CLOCKS_PER_SEC;
39     cout << "Время последовательного варианта " << time << endl;
40     cout << "Число вершин " << num_nodes << endl;
41 }
42
43 //2b
44 void node_process_parallel(struct node *tree)
45 {
46     // Подсчитываем число узлов
47     #pragma omp atomic
48     num_nodes++;
49     // Связываем с каждым узлом какую-то работу
50     // Работа имеет разную вычислительную сложность для различных вершин
51     work(tree->num);
52     // Выводим номер узла, который обработали
53     //cout << tree->num << " " << omp_get_thread_num() << endl;
54     #pragma omp task
55     if (tree->left)
56         node_process_parallel(tree->left);
57     #pragma omp task
58     if (tree->right)
59         node_process_parallel(tree->right);
60     return;
61 }
```

```
62
63
64 void task_2b(node * tree)
65 {
66     cout << "2a) Последовательный и параллельный варианты" << endl;
67     clock_t start, finish; // переменные для измерения времени
68     double time1, time2;
69     start = clock();
70     node_process(tree);
71     finish = clock();
72     time1 = (double)(finish - start)/CLOCKS_PER_SEC;
73
74     start = clock();
75     #pragma omp parallel
76     {
77         #pragma omp single
78         {
79             node_process_parallel(tree);
80         }
81     }
82     finish = clock();
83     time2 = (double)(finish - start)/CLOCKS_PER_SEC;
84     cout << "Время последовательного варианта " << time1 << endl;
85     cout << "Время параллельного варианта " << time2 << endl;
86 }
87
88
89 // 2c
90 void task_2c(node * tree)
91 {
92     cout << "2в) Параллельный вариант без 'pragma omp single'" << endl;
93     clock_t start, finish; // переменные для измерения времени
94     double time1, time2;
95     start = clock();
96     node_process(tree);
97     finish = clock();
98     time1 = (double)(finish - start)/CLOCKS_PER_SEC;
99     start = clock();
100    #pragma omp parallel
101    {
102        {
103            node_process_parallel(tree);
104        }
105    }
106    finish = clock();
107    time2 = (double)(finish - start)/CLOCKS_PER_SEC;
108    cout << "Время последовательного варианта " << time1 << endl;
109    cout << "Время параллельного варианта " << time2 << endl;
110 }
111
112
113 // 2d
114 void node_process_parallel_rising(struct node *tree)
115 {
116     // Подсчитываем число узлов
117     #pragma omp atomic
118     num_nodes++;
119
120     // Выводим номер узла, который обработали
121     //cout << tree->num << " " << omp_get_thread_num() << endl;
122     #pragma omp task
123     if (tree->left)
```

```
124         node_process_parallel(tree->left);
125     #pragma omp task
126     if (tree->right)
127         node_process_parallel(tree->right);
128 
129     // Связываем с каждым узлом какую-то работу
130     // Работа имеет разную вычислительную сложность для различных вершин
131     work(tree->num);
132 
133     return;
134 }
135 
136 
137 void task_2d(node * tree)
138 {
139     cout << "2г) Восходящий и нисходящий варианты" << endl;
140     clock_t start, finish; // переменные для измерения времени
141     double time1, time2;
142     start = clock();
143     cout << "Нисходящая обработка" << endl;
144     #pragma omp parallel
145     {
146         #pragma omp single
147         {
148             node_process_parallel(tree);
149         }
150     }
151     finish = clock();
152     time1 = (double)(finish - start)/CLOCKS_PER_SEC;
153 
154 
155     start = clock();
156     cout << "Восходящая обработка" << endl;
157     #pragma omp parallel
158     {
159         #pragma omp single
160         {
161             node_process_parallel_rising(tree);
162         }
163     }
164     finish = clock();
165     time2 = (double)(finish - start)/CLOCKS_PER_SEC;
166     cout << "Время нисходящей параллельной обработки " << time1 << endl;
167     cout << "Время восходящей параллельной обработки " << time2 << endl;
168 }
169 
170 #endif //TASKSOPENMP_2A_H
```

3.h

```
1 //
2 // Created by aleksey on 28.11.16.
3 //
4
5 #ifndef TASKSOPENMP_3_H
6 #define TASKSOPENMP_3_H
7
8 #include <iostream>
9 #include <ctime>
10 #include <omp.h>
11 #include "Tree.h"
12 using namespace std;
13
14 int par, prec;
15 int found_count;
16
17 //Обрабатываем узел
18 void find(struct node *tree)
19 {
20     // Подсчитываем число узлов
21     num_nodes++;
22     // Связываем с каждым узлом какую-то работу
23     // Работа имеет разную вычислительную сложность для различных вершин
24     //work(tree->num);
25     // Выводим номер узла, который обработали
26     //cout << tree->num << endl;
27     if (abs(tree->num - par) <= prec)
28         found_count += 1;
29
30     if (tree->left)
31         find(tree->left);
32     if (tree->right)
33         find(tree->right);
34     return;
35 }
36
37 //Обрабатываем узел
38 void find_parallel(struct node *tree)
39 {
40     // Подсчитываем число узлов
41     #pragma omp atomic
42     num_nodes++;
43     // Связываем с каждым узлом какую-то работу
44     // Работа имеет разную вычислительную сложность для различных вершин
45     // work(tree->num);
46     // Выводим номер узла, который обработали
47     //cout << tree->num << " " << omp_get_thread_num() << endl;
48     if (abs(tree->num - par) <= prec)
49         #pragma omp atomic
50         found_count += 1;
51
52     #pragma omp task
53     if (tree->left)
54         find_parallel(tree->left);
55     #pragma omp task
56     if (tree->right)
57         find_parallel(tree->right);
58     return;
59 }
60
61 void task_3a(node *tree, int par, int prec){
```

```
62      cout << "3a) Поиск элементов abs(N - " << par << ") < " << prec << endl;
63      ::par = par;
64      ::prec = prec;
65      found_count = 0;
66      num_nodes = 0;
67
68      clock_t start, finish; // переменные для измерения времени
69      double time1, time2;
70      start = clock();
71      find(tree);
72      finish = clock();
73      time1 = (double)(finish - start)/CLOCKS_PER_SEC;
74      cout << "Time serial is " << time1 << endl;
75      cout << "Число элементов (abs(N-par) <= prec) = " << found_count;
76      found_count = 0;
77      num_nodes = 0;
78
79      start = clock();
80      #pragma omp parallel
81      {
82      #pragma omp single
83          {
84              find_parallel(tree);
85          }
86      }
87      finish = clock();
88      time2 = (double)(finish - start)/CLOCKS_PER_SEC;
89
90      cout << "Время выполнения параллельного варианта" << time2 << endl;
91      cout << "Число вершин " << num_nodes << endl;
92      cout << "Число элементов (abs(N-par) <= prec) = " << found_count;
93  }
94
95
96  //Обрабатываем узел
97  void find_parallel_atomic(struct node *tree)
98  {
99      // Подсчитываем число узлов
100     #pragma omp atomic
101     num_nodes++;
102     // Связываем с каждым узлом какую-то работу
103     // Работа имеет разную вычислительную сложность для различных вершин
104     // work(tree->num);
105     // Выводим номер узла, который обработали
106     //cout << tree->num << " " << omp_get_thread_num() << endl;
107         if (abs(tree->num - par) <= prec)
108             #pragma omp atomic
109             found_count += 1;
110
111 #pragma omp task
112     if (tree->left)
113         find_parallel_atomic(tree->left);
114 #pragma omp task
115     if (tree->right)
116         find_parallel_atomic(tree->right);
117     return;
118 }
119
120 void task_3b(node *tree, int par, int prec){
121     cout << "3б) [Atomic] Поиск элементов abs(N - " << par << ") < " << prec <<
122 endl;
123     ::par = par;
```

```
124     ::prec = prec;
125     found_count = 0;
126     num_nodes = 0;
127 
128     clock_t start, finish; // переменные для измерения времени
129     double time1, time2;
130     start = clock();
131     find(tree);
132     finish = clock();
133     time1 = (double)(finish - start)/CLOCKS_PER_SEC;
134     cout << "Время посоледовательного варианта " << time1 << endl;
135     cout << "Число вершин (abs(N-par) <= prec) = " << found_count;
136     found_count = 0;
137     num_nodes = 0;
138 
139     start = clock();
140 #pragma omp parallel
141     {
142 #pragma omp single
143         {
144             find_parallel_atomic(tree);
145         }
146     }
147     finish = clock();
148     time2 = (double)(finish - start)/CLOCKS_PER_SEC;
149 
150     cout << "Время параллельного варианта (atomic) " << time2 << endl;
151     cout << "Число вершин " << num_nodes << endl;
152     cout << "Число вершин (abs(N-par) <= prec) = " << found_count;
153 }
154 
155 
156 int first_found_number = 0;
157 //Обрабатываем узел
158 void find_until_first(struct node *tree)
159 {
160     // Подсчитываем число узлов
161     #pragma omp atomic
162     num_nodes++;
163     // Связываем с каждым узлом какую-то работу
164     // Работа имеет разную вычислительную сложность для различных вершин
165     // work(tree->num);
166     // Выводим номер узла, который обработали
167     //cout << tree->num << " " << omp_get_thread_num() << endl;
168 
169     if (abs(tree->num - par) <= prec)
170     {
171         #pragma omp atomic
172         found_count += 1;
173         first_found_number = tree->num;
174         return;
175     }
176 
177 #pragma omp task
178     if (tree->left)
179         find_until_first(tree->left);
180 #pragma omp task
181     if (tree->right)
182         find_until_first(tree->right);
183     return;
184 }
185 
```

```
186 void task_3ge(node * tree, int par, int prec){
187     cout << "3ге) Последовательный и параллельный варианты поиска до 1
188 встретившейся вершины" << endl;
189     ::par = par;
190     ::prec = prec;
191     found_count = 0;
192     num_nodes = 0;
193     first_found_number = -1;
194 
195     clock_t start, finish; // переменные для измерения времени
196     double time1, time2;
197     start = clock();
198     find_until_first(tree);
199     finish = clock();
200     time1 = (double)(finish - start)/CLOCKS_PER_SEC;
201     cout << "Последовательный вариант " << time1 << endl;
202     cout << "Число вершин " << num_nodes << endl;
203     cout << "Первая найденная " << first_found_number << endl;
204     found_count = 0;
205     num_nodes = 0;
206     first_found_number = -1;
207 
208     start = clock();
209     #pragma omp parallel
210     {
211     #pragma omp single
212         {
213             find_until_first(tree);
214         }
215     }
216     finish = clock();
217     time2 = (double)(finish - start)/CLOCKS_PER_SEC;
218 
219     cout << "Параллельный вариант " << time2 << endl;
220     cout << "Число вершин " << num_nodes << endl;
221     cout << "Первая найденная " << first_found_number << endl;
222     found_count = 0;
223     num_nodes = 0;
224     first_found_number = -1;
    }
    #endif // TASKSOPENMP_3_H
```