МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

«Программирование многопоточных приложений. POSIX Threads»

студентки 2 курса, группы 21207

Черновской Яны Тихоновны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.Ю. Власенко

Новосибирск 2023

СОДЕРЖАНИЕ

ЦЕЛЬ

Освоить разработку многопоточных программ с использованием POSIX Threads API. Познакомиться с задачей динамического распределения работы между процессорами.

ЗАДАНИЕ

1. Написать программу с использованием POSIX Threads API, симмулирующую работу кластера.
2. Программа не должна зависеть от числа процессов.
3. Использование коммуникаций MPI между процессами.
4. Использование POSIX Threads для порождения нитей в рамках процессов.
5. Минимум в каждом процессе по 2 нити.
6. Вес заданий считать заранее неизвестным для процессов.

ОПИСАНИЕ РАБОТЫ

Есть список неделимых заданий, каждое из которых может быть выполнено независимо от другого. Задания могут иметь различный вычислительный вес, т.е. требовать при одних и тех же вычислительных ресурсах различного времени для выполнения. Считается, что этот вес нельзя узнать, пока задание не выполнено. После того, как все задания из списка выполнены, появляется новый список заданий. Необходимо организовать параллельную обработку заданий на нескольких компьютерах. Количество заданий существенно превосходит количество процессоров. Программа не должна зависеть от числа компьютеров.

Так как задания имеют различный вычислительный вес, а список обрабатывается итеративно, и требуется синхронизация перед каждой итерацией, то могут возникать ситуации, когда некоторые процессоры выполнили свою работу, а другие - еще нет. Если ничего не предпринять, первые будут простаивать в ожидании последних. Так возникает задача динамического распределения работы. Для ее решения на каждом процессоре заведем несколько потоков. Как минимум, потоков должно быть 2:

- поток, который обрабатывает задания и, когда задания закончились, обращается к другим компьютерам за добавкой к работе,
- поток, ожидающий запросов о работе от других компьютеров

Необходимо обеспечивать взаимное исключение потоков при добавлении заданий в список, удалении задач, выборке заданий для выполнения.

ЗАКЛЮЧЕНИЕ

Был получен вывод программы:

```
Процесс: 2. Конец итерации 0, время: 5.560453, результат: 244.163594, сделано заданий: 339
Процесс: 0. Конец итерации 0, время: 5.584907, результат: 340.377103, сделано заданий: 1137
Процесс: 1. Конец итерации 0, время: 5.597003, результат: 336.328635, сделано заданий: 544
Процесс: 3. Конец итерации 0, время: 5.619064, результат: 185.772823, сделано заданий: 206
Итерация = 0 Время дисбаланса: 0.058612
Процент дисбаланса: 1.043084
Процесс: 0. Конец итерации 1, время: 3.599489, результат: 321.158090, сделано заданий: 417
Процесс: 2. Конец итерации 1, время: 3.608285, результат: 309.188196, сделано заданий: 513
Процесс: 3. Конец итерации 1, время: 3.618121, результат: 186.289386, сделано заданий: 206
Процесс: 1. Конец итерации 1, время: 3.618307, результат: 285.278069, сделано заданий: 1029
Итерация = 1 Время дисбаланса: 0.018817
Процент дисбаланса: 0.520056
Процесс: 3. Конец итерации 2, время: 3.614452, результат: 368.262348, сделано заданий: 406
Процесс: 1. Конец итерации 2, время: 3.615120, результат: 367.640048, сделано заданий: 405
Процесс: 2. Конец итерации 2, время: 3.618254, результат: 185.667085, сделано заданий: 805
Процесс: 0. Конец итерации 2, время: 3.622767, результат: 185.470022, сделано заданий: 205
Итерация = 2 Время дисбаланса: 0.008315
Процент дисбаланса: 0.229509
Процесс: 2. Конец итерации 3, время: 5.399493, результат: 438.427749, сделано заданий: 562
Процесс: 3. Конец итерации 3, время: 5.412380, результат: 185.462176, сделано заданий: 807
Процесс: 1. Конец итерации 3, время: 5.413374, результат: 278.713247, сделано заданий: 306
Процесс: 0. Конец итерации 3, время: 5.428372, результат: 205.320854, сделано заданий: 263
Итерация = 3 Время дисбаланса: 0.028879
Процент дисбаланса: 0.532003
Процесс: 0. Конец итерации 4, время: 5.392630, результат: 324.932944, сделано заданий: 1139
Процесс: 2. Конец итерации 4, время: 5.405664, результат: 259.825788, сделано заданий: 338
Процесс: 1. Конец итерации 4, время: 5.421765, результат: 351.199692, сделано заданий: 571
Процесс: 3. Конец итерации 4, время: 5.431092, результат: 185.772823, сделано заданий: 206
Итерация = 4 Время дисбаланса: 0.038462
Процент дисбаланса: 0.708182
Процесс: 1. Конец итерации 5, время: 3.589489, результат: 297.585708, сделано заданий: 1058
Процесс: 2. Конец итерации 5, время: 3.601073, результат: 302.358853, сделано заданий: 527
Процесс: 0. Конец итерации 5, время: 3.616331, результат: 314.801898, сделано заданий: 415
Процесс: 3. Конец итерации 5, время: 3.619171, результат: 185.667085, сделано заданий: 206
Итерация = 5 Время дисбаланса: 0.029682
Процент дисбаланса: 0.820137
Процесс: 3. Конец итерации 6, время: 3.605334, результат: 326.837205, сделано заданий: 425
Процесс: 2. Конец итерации 6, время: 3.605465, результат: 324.853808, сделано заданий: 1109
Процесс: 1. Конец итерации 6, время: 3.617161, результат: 269.857596, сделано заданий: 434
Процесс: 0. Конец итерации 6, время: 3.632730, результат: 184.847722, сделано заданий: 205
Итерация = 6 Время дисбаланса: 0.027396
Процент дисбаланса: 0.754143
Процесс: 3. Конец итерации 7, время: 5.407023, результат: 188.623801, сделано заданий: 807
Процесс: 1. Конец итерации 7, время: 5.419040, результат: 277.504467, сделано заданий: 305
Процесс: 2. Конец итерации 7, время: 5.420957, результат: 438.092246, сделано заданий: 561
Процесс: 0. Конец итерации 7, время: 5.435029, результат: 202.216240, сделано заданий: 278
Итерация = 7 Время дисбаланса: 0.028007
Процент дисбаланса: 0.515297
Процесс = 0. Время: 36.407990
Процесс = 1. Время: 36.408929
Процесс = 2. Время: 36.407537
Процесс = 3. Время: 36.407682
```

Доля дисбаланса получилась достаточно маленькой, что сведетельствует о том, что программа работает корректно.

ПРИЛОЖЕНИЕ

Приложение 1. Полный листинг параллельной программы на C++

```
1.  #include <iostream>
2.  #include <mpi.h>
3.  #include <cmath>
4.  #include <unistd.h>
5.  #include <vector>
6.  #include <pthread.h>
7.  #include <cstdio>
8.
9.  struct Task {
10.     int repeat_num;
11. };
12.
13. const int THREAD_NUM = 2;
14. const int L = 10000;
15. const int ITERATIONS_COUNT = 8;
16.
17. pthread_mutex_t mutex;
18. pthread_t threads[THREAD_NUM];
19. std::vector<Task> tasks;
20. MPI_Datatype TASK_TYPE;
21.
22. int rank;
23. int size;
24. int iter_counter;
25. int task_index;
26.
27. int get_new_task(int proc_rank);
28. double calc_task(Task &task);
29. void generate_tasks(int count_tasks);
30.
31. void *calc_thread(__attribute__((unused)) void *args);
32. void *data_thread(__attribute__((unused)) void *args);
33. int run();
34.
35. int main(int argc, char **argv) {
36.     setlocale(LC_ALL, "Russian");
37.
38.     int provided;
39.     MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &provided);
40.
41.     MPI_Comm_size(MPI_COMM_WORLD, &size);
42.     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
43.
44.     if (provided != MPI_THREAD_MULTIPLE) {
```

```
45.             MPI_Finalize();
46.             return EXIT_FAILURE;
47.         }
48.
49.         /* Создание нового типа для отправки структуры Task _____*/
50.         int count = 1;
51.         int block_lengths[] = {1};
52.         MPI_Datatype types[] = {MPI_INT};
53.         MPI_Aint displs = 0;
54.         MPI_Type_create_struct(count, block_lengths, &displs, types,
55.                                &TASK_TYPE);
56.         MPI_Type_commit(&TASK_TYPE);
57.         /*-------------------------------------------------------------------------------------------*/
58.
59.         pthread_mutex_init(&mutex, nullptr);
60.         double start = MPI_Wtime();
61.
62.         if (run() != EXIT_SUCCESS) {
63.             pthread_mutex_destroy(&mutex);
64.             MPI_Type_free(&TASK_TYPE);
65.             MPI_Finalize();
66.             return EXIT_FAILURE;
67.         }
68.
69.         double end = MPI_Wtime();
70.         printf("Процесс = %d. Время: %lf\n", rank, end - start);
71.
72.         pthread_mutex_destroy(&mutex);
73.         MPI_Type_free(&TASK_TYPE);
74.         MPI_Finalize();
75.         return EXIT_SUCCESS;
76.     }
77.
78.     int run() {
79.         //атрибуты потока
80.         pthread_attr_t attrs;
81.
82.         //инициализация атрибутов потока
83.         if(pthread_attr_init(&attrs) != 0)
84.         {
85.             perror("Не могу инициализировать атрибуты");
86.             return EXIT_FAILURE;
87.         }
88.
89.         //установка атрибута "присоединенный"
90.         if(pthread_attr_setdetachstate(&attrs, PTHREAD_CREATE_JOINABLE) != 0)
91.         {
92.             perror("Ошибка в установке атрибута");
```

```
93.             abort();
94.         }
95.
96.         pthread_create(&threads[0], &attrs, calc_thread, nullptr);
97.         pthread_create(&threads[1], &attrs, data_thread, nullptr);
98.
99.         //освобождение ресурсов, занимаемых описателем атрибутов
100.        pthread_attr_destroy(&attrs);
101.
102.        for (auto & thread : threads) {
103.            if (pthread_join(thread, nullptr) != 0) {
104.                perror("Не могу подсоединиться к потоку");
105.                return EXIT_FAILURE;
106.            }
107.        }
108.        return EXIT_SUCCESS;
109.    }
110.
111.    void *calc_thread(__attribute__((unused)) void *args) {
112.        int *able_get_task_flags = new int[size];
113.        for (iter_counter = 0; iter_counter < ITERATIONS_COUNT; iter_counter++) {
114.            task_index = 0;
115.            double result = 0;
116.            int task_count = 0;
117.            int count = 100 * size;
118.
119.            generate_tasks(count);
120.            double start = MPI_Wtime();
121.
122.            for (int i = 0; i < size; i++) {
123.                able_get_task_flags[i] = 1;
124.            }
125.            able_get_task_flags[rank] = 0;
126.
127.            while (true) {
128.                while (task_index < tasks.size()) {
129.                    pthread_mutex_lock(&mutex);
130.                    Task task_to_do = tasks[task_index];
131.                    pthread_mutex_unlock(&mutex);
132.
133.                    result += calc_task(task_to_do);
134.
135.                    task_count++;
136.                    task_index++;
137.                }
138.
139.                int i = 0;
140.                for (; i < size;) {
```

```
141.                 if (!able_get_task_flags[i]) {
142.                     i++;
143.                     continue;
144.                 } else {
145.                     task_count++;
146.                     able_get_task_flags[i] = get_new_task(i);
147.                     break;
148.                 }
149.             }
150.             if (i == size) {
151.                 break;
152.             }
153.         }
154.
155.         double end = MPI_Wtime();
156.         double time = end - start;
157.         printf("Процесс: %d. Конец итерации %d, время: %lf, "
158.                "результат: %lf, сделано заданий: %d\n", rank, iter_counter, time, result,
    task_count);
159.
160.         MPI_Barrier(MPI_COMM_WORLD);
161.
162.         double max_time = 0;
163.         double min_time = 0;
164.         MPI_Reduce(&time, &max_time, 1, MPI_DOUBLE, MPI_MAX, 0,
165.                    MPI_COMM_WORLD);
166.         MPI_Reduce(&time, &min_time, 1, MPI_DOUBLE, MPI_MIN, 0,
167.                    MPI_COMM_WORLD);
168.
169.         if(rank == 0) {
170.             printf("Итерация = %d Время дисбаланса: %lf\n"
171.                    "Процент дисбаланса: %lf\n",
172.                    iter_counter, max_time - min_time, (max_time - min_time) / max_time * 100.0);
173.         }
174.     }
175.
176.     int request = 0;
177.
178.     MPI_Send(&request, 1, MPI_INT, rank, 1, MPI_COMM_WORLD);
179.
180.     delete[] able_get_task_flags;
181.     return nullptr;
182. }
183.
184. void *data_thread(__attribute__((unused)) void *args) {
185.     while (iter_counter < ITERATIONS_COUNT) {
186.         MPI_Status status;
187.         int res;
```

```
188.
189.         MPI_Recv(&res, 1, MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD,
190.                 &status);
191.
192.         if (res == 0) break;
193.     pthread_mutex_lock(&mutex);
194.         if (task_index >= tasks.size()) {
195.     pthread_mutex_unlock(&mutex);
196.             int answer = 0;
197.             MPI_Send(&answer, 1, MPI_INT, status.MPI_SOURCE, 2,
198.                     MPI_COMM_WORLD);
199.         } else {
200.
201.             Task task_to_send = tasks.back();
202.             tasks.pop_back();
203.             pthread_mutex_unlock(&mutex);
204.
205.             int answer = 1;
206.             MPI_Send(&answer, 1, MPI_INT, status.MPI_SOURCE, 2,
207.                     MPI_COMM_WORLD);
208.             MPI_Send(&task_to_send, 1, TASK_TYPE, status.MPI_SOURCE, 3,
209.                     MPI_COMM_WORLD);
210.
211.         }
212.     }
213.     return nullptr;
214. }
215.
216. int get_new_task(int proc_rank) {
217.
218.     int request = 1;
219.
220.     MPI_Send(&request, 1, MPI_INT, proc_rank, 1, MPI_COMM_WORLD);
221.     MPI_Recv(&request, 1, MPI_INT, proc_rank, 2, MPI_COMM_WORLD,
222.             MPI_STATUS_IGNORE);
223.
224.     if (request == 0) return 0;
225.
226.     Task task{};
227.     MPI_Recv(&task, 1, TASK_TYPE, proc_rank, 3, MPI_COMM_WORLD,
228.             MPI_STATUS_IGNORE);
229.     pthread_mutex_lock(&mutex);
230.     tasks.push_back(task);
231.     pthread_mutex_unlock(&mutex);
232.
233.     return 1;
234. }
235.
```

```
236.  double calc_task(Task &task) {
237.      double res = 0;
238.      for (int i = 0; i < task.repeat_num; i++) {
239.          res += sin(i);
240.      }
241.
242.      return res;
243.  }
244.
245.  void generate_tasks(int count_tasks)
246.  {
247.      pthread_mutex_lock(&mutex);
248.      tasks.clear();
249.      for (int i = 0; i < count_tasks; i++) {
250.          Task task{};
251.          task.repeat_num = std::abs(50 - i % 100) * std::abs(rank - (iter_counter % size)) * L;
252.          tasks.push_back(task);
253.      }
254.      pthread_mutex_unlock(&mutex);
255.  }
256.
```