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 ЛАБОРАТОРНОЙ РАБОТЫ

«Программирование многопоточных приложений. POSIX Threads»

студентки 2 курса, 19202 группы

Хаецкой Дарьи Владимировны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
А.Ю. Власенко

Новосибирск 2021

ЗАДАНИЕ

Реализовать некоторый аналог вычислительного кластера, принимающего при старте некоторый список задач для каждой “ноды кластера” - процесса MPI программы.

Реализовать систему автобалансировки задач между всеми нодами в случае, когда у какой-то из доступных закончились собственные,т.е. в тот момент, когда вычислительная эффективность кластера начинает уменьшаться. Общение нод, т.е. процессов, реализовать посредством стандартного MPI.
Работа ноды кластера должна быть реализована следующим образом: в процессе должно быть запущено 2 POSIX потока:

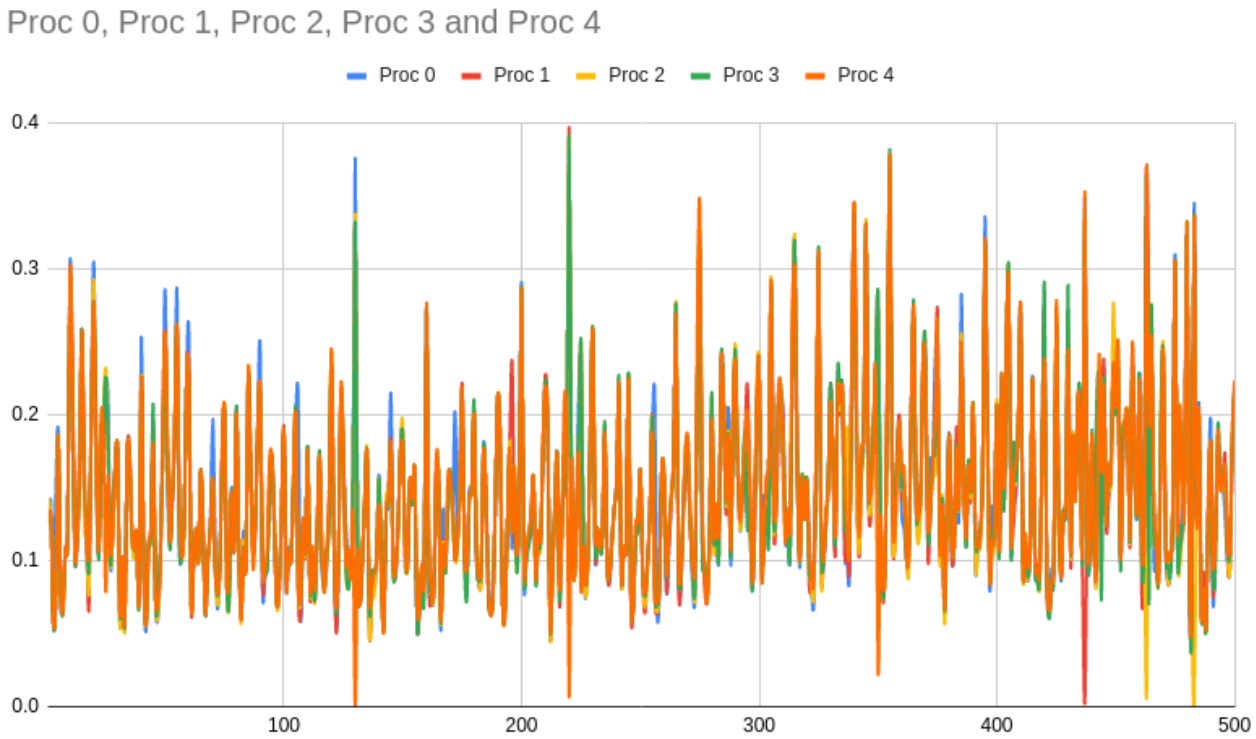
- поток “исполнитель”, исполняющий поступившие к нему задачи;
- поток “коммуникатор”, коммуницирующий со всеми остальными нодами.

ОПИСАНИЕ РАБОТЫ

1. В каждом процессе создается два потока: вычислительный и управляющий.
   Управляющий поток исполняет `ReceiverStartRoutine`, а вычислительный - `ExecutorStartRoutine`.
2. Вычислительный поток исполняет свой набор заданий, после чего опрашивает остальный процессы на предмет оставшихся заданий, если таковых не имеется, он получает от опрашиваемого процесса 0, иначе, управляющий процесс получает сначала количество заданий, а затем сами задания.
3. Управляющий процесс же в свою очередь отвечает на аналогичные запросы от других процессов, следит за числом оставшихся заданий процесса и если оно больше определённого числа, отдаёт часть заданий запрашивающему процессу.
4. Для синхронизации используются `pthread_mutex`. Мьютекс блокируется при извлечении задания из массива задач и при передаче и приёме заданий от других процессов

Анализ времени работы и разбалансировки.

В ходе измерения времени работы программы были получены следующие данные:



По оси OX отложены итерации, а по оси OУ - время работы в секундах. На графике видно, что в общей тенденции все процессы работают примерно одинаковое время.

Пример консольного вывода программы

```
Process 2 requested tasks. I have 0 tasks now.
Process 4 answered -565
Process 0 executed 222 tasks. 0 were additional.
Process 3 executed 703 tasks. 2 were additional.
Cos sum is 6.64197e+09. Time taken: 0.187363
Max time difference: 0.00212039
Disbalance rate is 1.13137%
Proc 3 finished iterations sending signal
Process 2 executed 360 tasks. 20 were additional.
Cos sum is 6.50617e+09. Time taken: 0.187417
Max time difference: 0.00212039
Disbalance rate is 1.13137%
Proc 2 finished iterations sending signal
Process 4 requested tasks. I have 0 tasks now.
Process 4 executed 2201 tasks. 0 were additional.
Cos sum is 6.94822e+09. Time taken: 0.187347
Max time difference: 0.00212039
Disbalance rate is 1.13137%
Proc 4 finished iterations sending signal
Process 2 requested tasks. I have 0 tasks now.
Process 2 requested tasks. I have 0 tasks now.
Process 1 executed 322 tasks. 2 were additional.
Cos sum is 6.59347e+09. Time taken: 0.185297
Max time difference: 0.00212039
Disbalance rate is 1.13137%
Cos sum is 7.00226e+09. Time taken: 0.185654
Max time difference: 0.00212039
Disbalance rate is 1.13137%
Proc 0 finished iterations sending signal
Process 3 requested tasks. I have 0 tasks now.
Proc 1 finished iterations sending signal
Process 3 requested tasks. I have 0 tasks now.
Summary disbalance:13.0216%
time taken: 69.466
(base) rey@zen-ux533fd:~/CLionProjects/Lab5Cluster$
```

Приложение 1. *Листинг программы main.cpp*

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
#include <cmath>
#include <mpi.h>
#include <cstdlib>
#include <fstream>
#include "utils.h"


#define L 1000
#define LISTS_COUNT 500
#define TASK_COUNT 2000
#define MIN_TASKS_TO_SHARE 2

pthread_t threads[2];
pthread_mutex_t mutex;
int *tasks;
std::ofstream *LogFiles;

double SummaryDisbalance = 0;
bool FinishedExecution = false;

int ProcessCount;
int ProcessRank;
int RemainingTasks;
int ExecutedTasks;
int AdditionalTasks;
double globalRes = 0;

void printTasks(int *taskSet){
   std::cout << ANSI_RED << "Process :" << ProcessRank;
   for (int i = 0; i < TASK_COUNT; i++){
       std::cout << taskSet[i] << " ";
   }
   std::cout << ANSI_RESET << std::endl;
}

void initializeTaskSet(int *taskSet, int taskCount, int iterCounter){
   for (int i = 0; i < taskCount; i++){
               taskSet[i] = abs(50 - i%100)*abs(ProcessRank - (iterCounter % 
ProcessCount))*L;
   }
}

void executeTaskSet(int *taskSet){
   for(int i = 0; i < RemainingTasks; i++){
       pthread_mutex_lock(&mutex);
       int weight = taskSet[i];
       pthread_mutex_unlock(&mutex);

       for(int j = 0; j < weight; j++){
           globalRes += cos(0.001488);
       }

       ExecutedTasks++;
```

```
        }
        RemainingTasks = 0;
     }

     void* ExecutorStartRoutine(void* args){
        tasks = new int[TASK_COUNT];
              double   StartTime,   FinishTime,   IterationDuration,   ShortestIteration,
LongestIteration;

        for(int i = 0; i < LISTS_COUNT; i++){
            StartTime = MPI_Wtime();
            MPI_Barrier(MPI_COMM_WORLD);
              std::cout << ANSI_RED << "Iteration " << i << ". Initializing tasks. " <<
ANSI_RESET << std::endl;
            initializeTaskSet(tasks, TASK_COUNT, i);
            ExecutedTasks = 0;
            RemainingTasks = TASK_COUNT;
            AdditionalTasks = 0;

            executeTaskSet(tasks);
             std::cout << ANSI_BLUE << "Process " << ProcessRank << " executed tasks in "
<<
                        MPI_Wtime() - StartTime << " Now requesting for some additional. "
<< ANSI_RESET << std::endl;
            int ThreadResponse;

            for (int procIdx = 0; procIdx < ProcessCount; procIdx++){

                if (procIdx != ProcessRank){
                     std::cout << ANSI_WHITE << "Process " << ProcessRank << " is asking "
<< procIdx <<
                    " for some tasks."<< ANSI_RESET << std::endl;
                    MPI_Send(&ProcessRank, 1, MPI_INT, procIdx, 888, MPI_COMM_WORLD);
                     std::cout << ANSI_PURPLE << "waiting for task count" << ANSI_RESET <<
std::endl;
                        MPI_Recv(&ThreadResponse, 1, MPI_INT, procIdx, SENDING_TASK_COUNT,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                      std::cout << ANSI_WHITE << "Process " << procIdx << " answered " <<
ThreadResponse << ANSI_RESET << std::endl;

                    if (ThreadResponse != NO_TASKS_TO_SHARE){
                        AdditionalTasks = ThreadResponse;
                        memset(tasks, 0, TASK_COUNT);
                          std::cout << ANSI_PURPLE << "waiting for tasks" << ANSI_RESET <<
std::endl;
                         MPI_Recv(tasks, AdditionalTasks, MPI_INT, procIdx, SENDING_TASKS,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                        pthread_mutex_lock(&mutex);
                        RemainingTasks = AdditionalTasks;
                        pthread_mutex_unlock(&mutex);
                        executeTaskSet(tasks);
                    }
                }

            }
            FinishTime = MPI_Wtime();
            IterationDuration = FinishTime - StartTime;

             MPI_Allreduce(&IterationDuration, &LongestIteration, 1, MPI_DOUBLE, MPI_MAX,
MPI_COMM_WORLD);
```

```
            MPI_Allreduce(&IterationDuration, &ShortestIteration, 1, MPI_DOUBLE, MPI_MIN,
MPI_COMM_WORLD);

            MPI_Barrier(MPI_COMM_WORLD);

               std::cout << ANSI_GREEN << "Process " << ProcessRank << " executed " <<
ExecutedTasks <<
                " tasks. " << AdditionalTasks << " were additional." << ANSI_RESET <<
std::endl;
             std::cout << ANSI_CYAN << "Cos sum is " << globalRes << ". Time taken: " <<
IterationDuration << std::endl;
            SummaryDisbalance += (LongestIteration - ShortestIteration)/LongestIteration;
             std::cout << "Max time difference: " << LongestIteration - ShortestIteration
<< ANSI_RESET << std::endl;
            std::cout << ANSI_PURPLE_BG << "Disbalance rate is " <<
              ((LongestIteration - ShortestIteration)/ LongestIteration) * 100 << "%" <<
ANSI_RESET << std::endl;
            LogFiles[ProcessRank] << IterationDuration << std::endl;
        }

         std::cout << ANSI_RED << "Proc " << ProcessRank << " finished iterations sending
signal" << ANSI_RESET << std::endl;
        pthread_mutex_lock(&mutex);
        FinishedExecution = true;
        pthread_mutex_unlock(&mutex);
        int Signal = EXECUTOR_FINISHED_WORK;
        MPI_Send(&Signal, 1, MPI_INT, ProcessRank, 888, MPI_COMM_WORLD);
        delete [] tasks;
        pthread_exit(nullptr);
     }

     void* ReceiverStartRoutine(void* args){
        int AskingProcRank, Answer, PendingMessage;
        MPI_Status status;
        MPI_Barrier(MPI_COMM_WORLD);
        while (!FinishedExecution){
               MPI_Recv(&PendingMessage, 1, MPI_INT, MPI_ANY_SOURCE, 888, MPI_COMM_WORLD,
&status);

            if (PendingMessage == EXECUTOR_FINISHED_WORK){
                  std::cout << ANSI_RED << "Executor finished work on proc " << ProcessRank
<< ANSI_RESET << std::endl;
            }
            AskingProcRank = PendingMessage;
            pthread_mutex_lock(&mutex);
             std::cout << ANSI_YELLOW << "Process " << AskingProcRank << " requested tasks.
I have " <<
            RemainingTasks << " tasks now. " << ANSI_RESET << std::endl;
            if (RemainingTasks >= MIN_TASKS_TO_SHARE){
                Answer = RemainingTasks / (ProcessCount*2);
                RemainingTasks = RemainingTasks / (ProcessCount*2);

                    std::cout << ANSI_PURPLE << "Sharing " << Answer << " tasks. " <<
ANSI_RESET << std::endl;

                       MPI_Send(&Answer, 1, MPI_INT, AskingProcRank, SENDING_TASK_COUNT,
MPI_COMM_WORLD);
                  MPI_Send(&tasks[TASK_COUNT - Answer], Answer, MPI_INT, AskingProcRank,
SENDING_TASKS, MPI_COMM_WORLD);
            } else {
                Answer = NO_TASKS_TO_SHARE;
```

```
                        MPI_Send(&Answer, 1, MPI_INT, AskingProcRank, SENDING_TASK_COUNT,
MPI_COMM_WORLD);
            }
            pthread_mutex_unlock(&mutex);
        }

        pthread_exit(nullptr);
     }


     int main(int argc, char* argv[]) {
        int ThreadSupport;
        MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &ThreadSupport);
        if(ThreadSupport != MPI_THREAD_MULTIPLE){
            MPI_Finalize();
            return -1;
        }

        MPI_Comm_rank(MPI_COMM_WORLD, &ProcessRank);
        MPI_Comm_size(MPI_COMM_WORLD, &ProcessCount);

        pthread_mutex_init(&mutex, nullptr);
        pthread_attr_t ThreadAttributes;

        LogFiles = new std::ofstream [ProcessCount];
        char* name = new char[12];
        for (int i = 0; i < ProcessCount; i++) {
            sprintf(name, "Log_%d.txt", i);
            LogFiles[i].open(name);
        }
        double start = MPI_Wtime();
        pthread_attr_init(&ThreadAttributes);
        pthread_attr_setdetachstate(&ThreadAttributes, PTHREAD_CREATE_JOINABLE);
        pthread_create(&threads[0], &ThreadAttributes, ReceiverStartRoutine, NULL);
        pthread_create(&threads[1], &ThreadAttributes, ExecutorStartRoutine, NULL);
        pthread_join(threads[0], nullptr);
        pthread_join(threads[1], nullptr);
        pthread_attr_destroy(&ThreadAttributes);
        pthread_mutex_destroy(&mutex);

        if (ProcessRank == 0){
                            std::cout   <<   ANSI_GREEN   <<   "Summary   disbalance:"   <<
SummaryDisbalance/(LISTS_COUNT)*100 << "%" << ANSI_GREEN << std::endl;
             std::cout << ANSI_GREEN << "time taken: " << MPI_Wtime() - start << ANSI_GREEN
<< std::endl;
        }

        MPI_Finalize();
        return 0;
     }

     }
```

Приложение 2. *Листинг программы utils.cpp*

```
//
// Created by daria on 5/15/21.
//

#ifndef LAB5CLUSTER_UTILS_H
#define LAB5CLUSTER_UTILS_H

#define EXECUTOR_FINISHED_WORK -1
#define SENDING_TASKS 666
#define SENDING_TASK_COUNT 787
#define NO_TASKS_TO_SHARE -565

#define ANSI_RESET "\033[0m"
#define ANSI_BLACK "\033[30m"
#define ANSI_RED "\033[31m"
#define ANSI_GREEN "\033[32m"
#define ANSI_YELLOW "\033[33m"
#define ANSI_BLUE "\033[34m"
#define ANSI_PURPLE "\033[35m"
#define ANSI_CYAN "\033[36m"
#define ANSI_WHITE "\033[37m"
#define ANSI_PURPLE_BG "\033[45m\033[30m"

#endif //LAB5CLUSTER_UTILS_H
```

Вывод

В ходе выполнения данной лабораторной работы мной были освоены концепции POSIX Threads, научилась динамически распределять работу между процессами и разделение ответственности между потоками.