Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное

образовательное учреждение высшего образования

«Южно-Уральский государственный университет

(национальный исследовательский университет)»

Высшая школы электроники и компьютерных наук

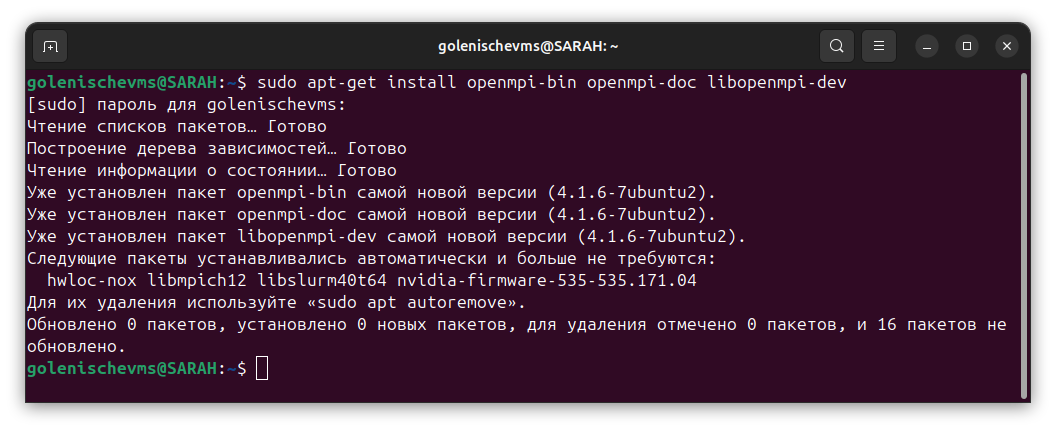
Кафедра системного программирования

ОТЧЕТ  
о лабораторной работе №4  
по дисциплине «Технологии параллельного программирования»

Выполнил:   
студент группы КЭ-220   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Голенищев А. Б.   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.   
   
Отчет принял:   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Жулев А. Э.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

***Задание 10. Создание проекта в среде MS Visual Studio с поддержкой MPI***

Будем использовать привычную среду QtCreator. Продемонстрован процесс настройки проекта с MPI в ней. Установим поддержку MPI компиляторами,   
рисунок 1.

Рисунок 1. Установка MPI в ОС Ubuntu 22.04 LTS

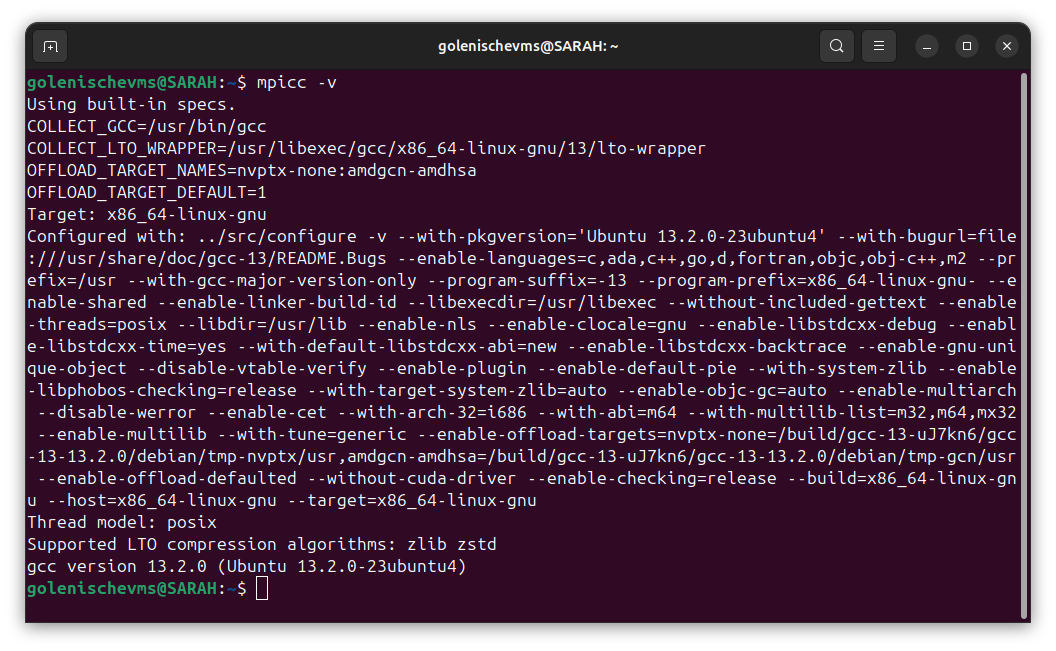
Проверим установку, рисунок 2.

Рисунок 2. Проверка установки OpenMPI

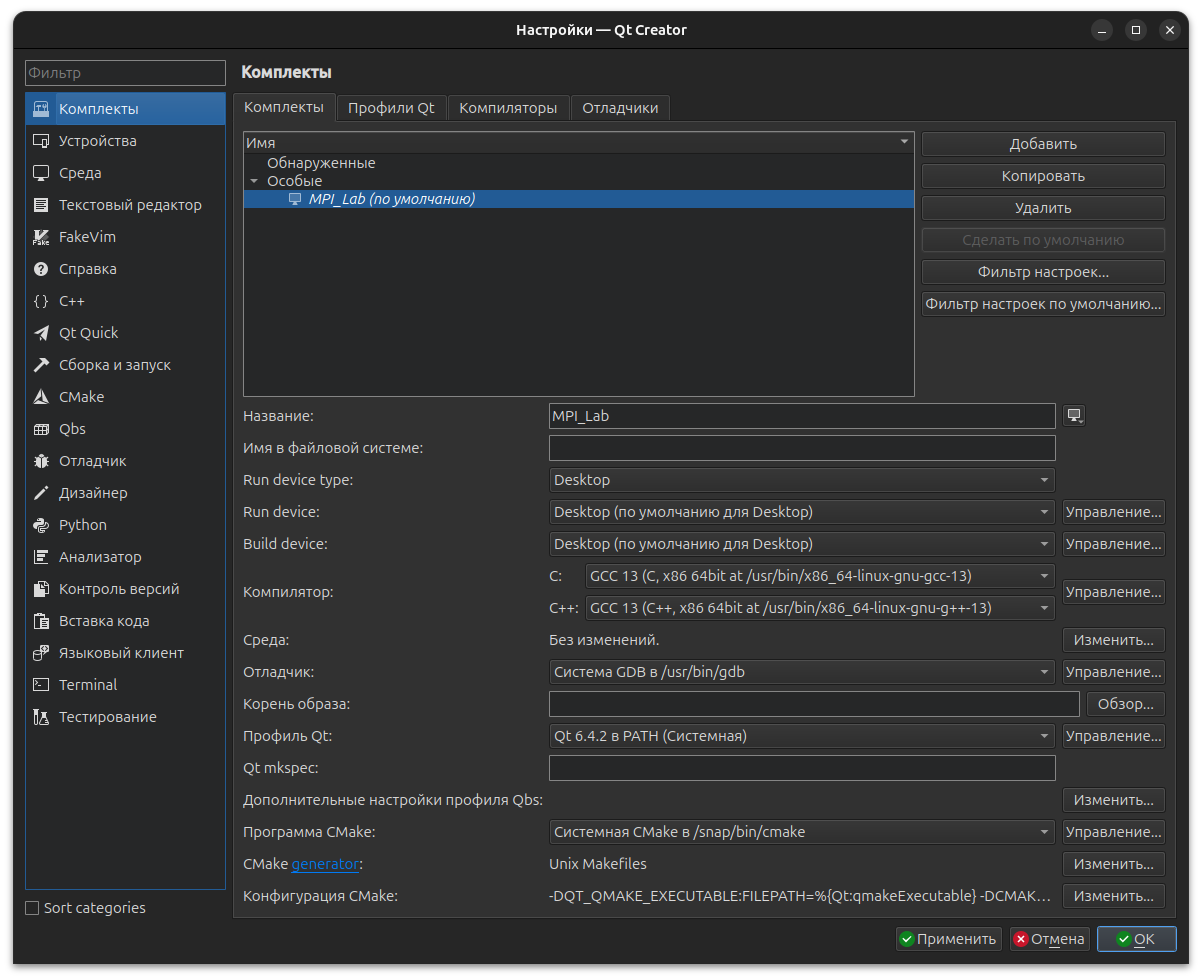
Теперь создадим проект. Настроим комплект для запуска приложения, рисунок 3. Настроим проект, листнинг 1.

Рисунок 3. Настройка комплекта для запуска приложения

TEMPLATE = app

CONFIG += console c++17

CONFIG -= app\_bundle

CONFIG -= qt

# Указываем компилятор MPI

QMAKE\_CC = mpicc

QMAKE\_CXX = mpic++

# Путь к заголовочным файлам MPI

INCLUDEPATH += /usr/include/openmpi

# Библиотеки MPI

LIBS += -lmpi\_cxx -lmpi -lpthread -lrt

# Дополнительные флаги компиляции

QMAKE\_CXXFLAGS += -Bsymbolic-functions

# Основной файл проекта

SOURCES += \

main.cpp

Листнинг 1. Настройка проекта QT с MPI

В QtCreator создадим проект консольного приложения на языке C++. По умолчанию будет создан main.cpp, содержащий код программы для проверки работы MPI, листнинг 2. Проведем сборку, а затем запустим приложение с помощью MPI,   
рисунок 4.

#include <stdio.h>

#include <mpi.h>

int main (int argc, char\* argv[])

{

MPI\_Init(&argc, &argv);

printf("MPI Process\n");

MPI\_Finalize();

return 0;

}

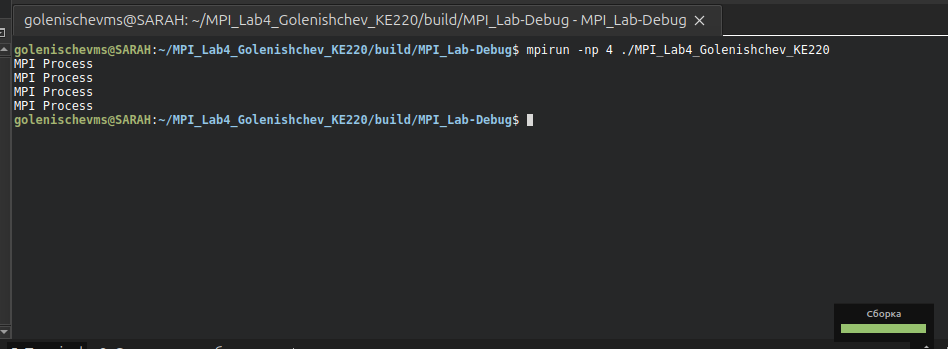
Листнинг 2. Код программы проверочной программы

Рисунок 4. Запуск собранного исполняемого файла с MPI

***Задание 11. Программа «I am!»***

Напишем программу, которая будет выводить номер каждой нити в OpenMPI, листнинг 3. Продемонстрирована работа программы, рисунок 5.

#include <mpi.h>

#include <stdio.h>

// Golenishchev Artem, KE-220 Task 11

int main(int argc, char \*argv[])

{

MPI\_Init(&argc, &argv);

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

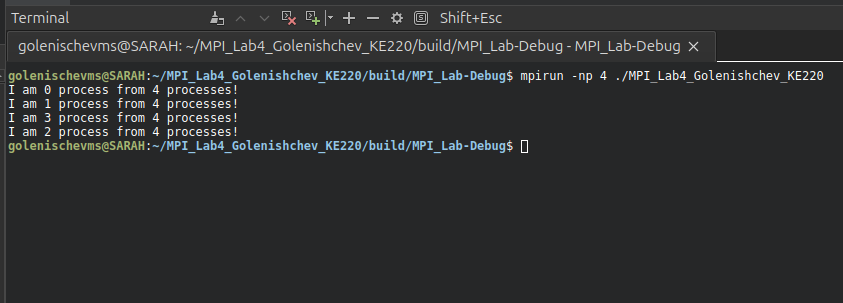
printf("I am %d process from %d processes!\n", rank, size);

MPI\_Finalize();

return 0;

}

Листнинг 3. Код программы «I am!»

Рисунок 5. Результат выполнения программы «I am!» в QtCreator

Программа использует MPI для распределения процессов, где каждый процесс выводит свой уникальный номер (rank) и общее количество процессов (size) в формате: "I am <номер> process from <количество> processes!". Синхронизация между процессами осуществляется через MPI\_Barrier, а завершение работы MPI происходит с вызовом MPI\_Finalize.

***Задание 12. Программа «На первый-второй рассчитайся!»***

Напишем программу, которая будет определять четность номеров процессов в OpenMPI, листнинг 4. Продемонстрирована работа программы, рисунок 6.

#include <mpi.h>

#include <stdio.h>

// Golenishchev Artem, KE-220 Task 12

int main(int argc, char \*argv[]) {

MPI\_Init(&argc, &argv);

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if (rank == 0) {

printf("%d processes.\n", size);

} else if (rank % 2 == 0) {

printf("I am %d process: SECOND!\n", rank);

} else {

printf("I am %d process: FIRST!\n", rank);

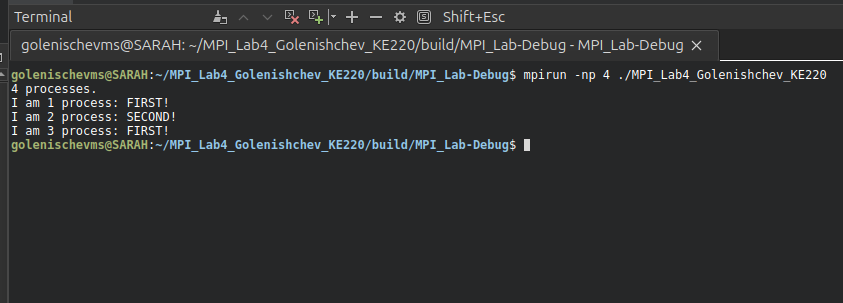
}

MPI\_Finalize();

return 0;

}

Листнинг 4. Код программы для определения четности номеров

Рисунок 6. Демонстрация работы программы определения четности потоков

***Ответы на вопросы:***

1. Что такое MPI? Какую модель параллельного программирования он реализует, на какую архитектуру ориентирован? Как Вы подключили его в используемой системе программирования?

MPI (Message Passing Interface) — это стандарт для организации параллельного программирования, основанный на модели передачи сообщений. Он ориентирован на распределенные системы с общей или раздельной памятью, например, кластеры. MPI реализует взаимодействие между процессами через обмен сообщениями. В системе программирования MPI подключается с помощью библиотеки mpi.h, которая предоставляет необходимые функции, такие как MPI\_Init, MPI\_Comm\_rank, MPI\_Comm\_size и другие.

1. Чем в MPI реализованы параллельно выполняемые подзадачи? Как и в какой момент они запускаются? До каких пор существуют? Чем идентифицируются?

В MPI параллельно выполняемые подзадачи представлены отдельными процессами, которые запускаются при старте программы через вызов mpirun или mpiexec. Они создаются и инициализируются функцией MPI\_Init() и существуют до вызова MPI\_Finalize(). Каждый процесс идентифицируется уникальным номером, называемым рангом, который возвращается функцией MPI\_Comm\_rank().

1. Что такое коммуникатор? Как учитываются входящие в него процессы?

Коммуникатор в MPI — это группа процессов, которые могут взаимодействовать друг с другом. Главный коммуникатор MPI\_COMM\_WORLD включает все процессы программы и используется по умолчанию. Входящие в коммуникатор процессы определяются при его создании, а их количество можно узнать с помощью функции MPI\_Comm\_size().

1. Что делают функции MPI\_Init() и MPI\_Finalize()? Какую роль играют? Сколько раз могут быть вызваны?

Функция MPI\_Init() инициализирует среду MPI и подготавливает все процессы к выполнению параллельной программы, а MPI\_Finalize() завершает работу MPI, освобождая все связанные ресурсы. Эти функции могут быть вызваны ровно один раз за выполнение программы: MPI\_Init() в начале и MPI\_Finalize() в конце.

1. Для чего любой параллельной программе нужна информация о количестве запущенных процессов(потоков) и идентификатор текущего объекта?

Информация о количестве процессов и идентификаторе текущего процесса необходима для распределения задач между процессами, синхронизации их работы и правильного маршрута обмена данными. Это позволяет каждому процессу знать свою роль в вычислительном процессе и эффективно взаимодействовать с другими, избегая конфликтов и дублирования.

***Выводы:***

Изучили основные принципы работы с MPI (Message Passing Interface) для реализации параллельных программ. Были освоены процессы инициализации и завершения MPI-приложений, использование функций для определения ранга процессов и их количества, а также применение коммуникаторов для организации взаимодействия между процессами. Были разработаны и протестированы программы, демонстрирующие базовые возможности MPI: вывод информации о процессах и распределение задач между ними. Работа показала важность синхронизации и четкого распределения задач для эффективного выполнения параллельных вычислений.