

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»

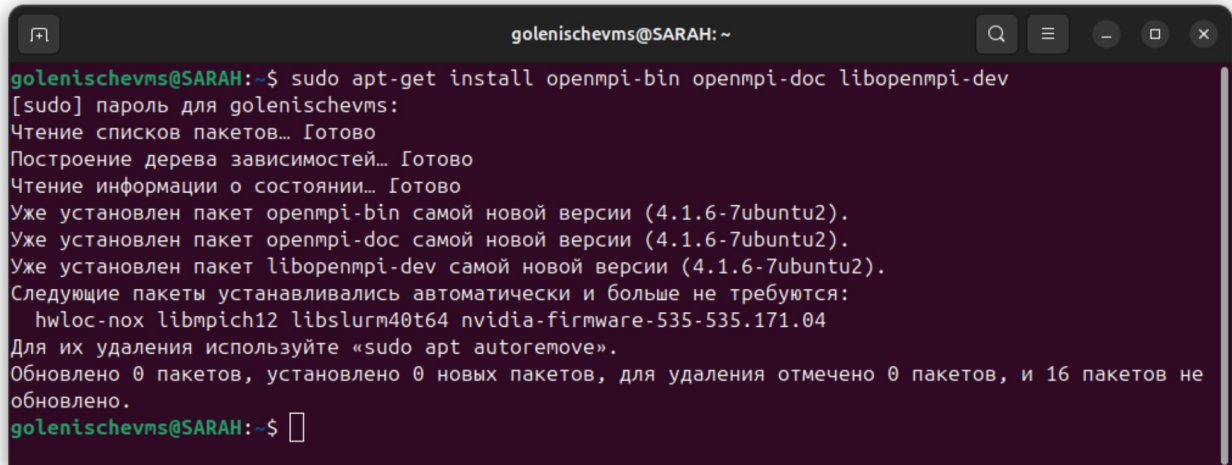
Выполнил:  
студент группы КЭ-220  
\_\_\_\_\_/Голенищев А. Б.  
\_\_\_\_\_ 2024 г.

Отчет принял:  
\_\_\_\_\_/Жулев А. Э.  
\_\_\_\_\_ 2024 г.

Челябинск  
2024

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

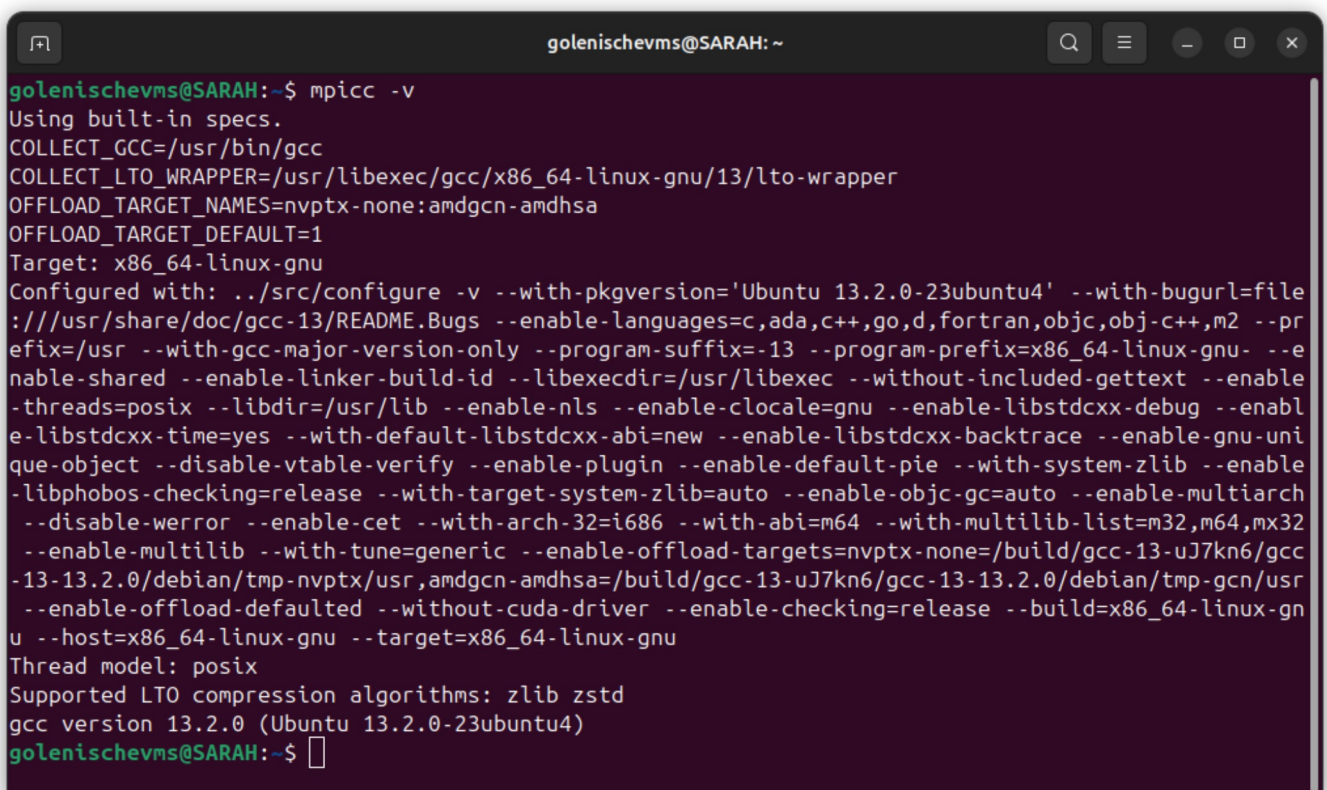
Будем использовать привычную среду QtCreator. Продемонстрирован процесс настройки проекта с MPI в ней. Установим поддержку MPI компиляторами, рисунок 1.



```
golenischevms@SARAH: ~  
golenischevms@SARAH:~$ sudo apt-get install openmpi-bin openmpi-doc libopenmpi-dev  
[sudo] пароль для golenischevms:  
Чтение списков пакетов... Готово  
Построение дерева зависимостей... Готово  
Чтение информации о состоянии... Готово  
Уже установлен пакет openmpi-bin самой новой версии (4.1.6-7ubuntu2).  
Уже установлен пакет openmpi-doc самой новой версии (4.1.6-7ubuntu2).  
Уже установлен пакет libopenmpi-dev самой новой версии (4.1.6-7ubuntu2).  
Следующие пакеты устанавливались автоматически и больше не требуются:  
  hwloc-nox libmpich12 libslurm40t64 nvidia-firmware-535-535.171.04  
Для их удаления используйте «sudo apt autoremove».  
Обновлено 0 пакетов, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 16 пакетов не  
обновлено.  
golenischevms@SARAH:~$
```

Рисунок 1. Установка MPI в ОС Ubuntu 22.04 LTS

Проверим установку, рисунок 2.



```
golenischevms@SARAH:~$ mpicc -v  
Using built-in specs.  
COLLECT_GCC=/usr/bin/gcc  
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-linux-gnu/13/lto-wrapper  
OFFLOAD_TARGET_NAMES=nvptx-none:amdgc-n-amdhsa  
OFFLOAD_TARGET_DEFAULT=1  
Target: x86_64-linux-gnu  
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 13.2.0-23ubuntu4' --with-bugurl=file  
:///usr/share/doc/gcc-13/README.Bugs --enable-languages=c,ada,c++,go,d,fortran,objc,obj-c++,m2 --pr  
efix=/usr --with-gcc-major-version-only --program-suffix=-13 --program-prefix=x86_64-linux-gnu- --e  
nable-shared --enable-linker-build-id --libexecdir=/usr/libexec --without-included-gettext --enable  
-threads=posix --libdir=/usr/lib --enable-nls --enable-clocale=gnu --enable-libstdcxx-debug --enabl  
e-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-libstdcxx-backtrace --enable-gnu-uni  
que-object --disable-vtable-verify --enable-plugin --enable-default-pie --with-system-zlib --enable  
-libphobos-checking=release --with-target-system-zlib=auto --enable-objc-gc=auto --enable-multiarch  
--disable-werror --enable-cet --with-arch=32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32  
--enable-multilib --with-tune=generic --enable-offload-targets=nvptx-none=/build/gcc-13-uJ7kn6/gcc  
-13-13.2.0/debian/tmp-nvptx/usr,amdgc-n-amdhsa=/build/gcc-13-uJ7kn6/gcc-13-13.2.0/debian/tmp-gcn/usr  
--enable-offload-defaulted --without-cuda-driver --enable-checking=release --build=x86_64-linux-gn  
u --host=x86_64-linux-gnu --target=x86_64-linux-gnu  
Thread model: posix  
Supported LTO compression algorithms: zlib zstd  
gcc version 13.2.0 (Ubuntu 13.2.0-23ubuntu4)  
golenischevms@SARAH:~$
```

Рисунок 2. Проверка установки OpenMPI

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

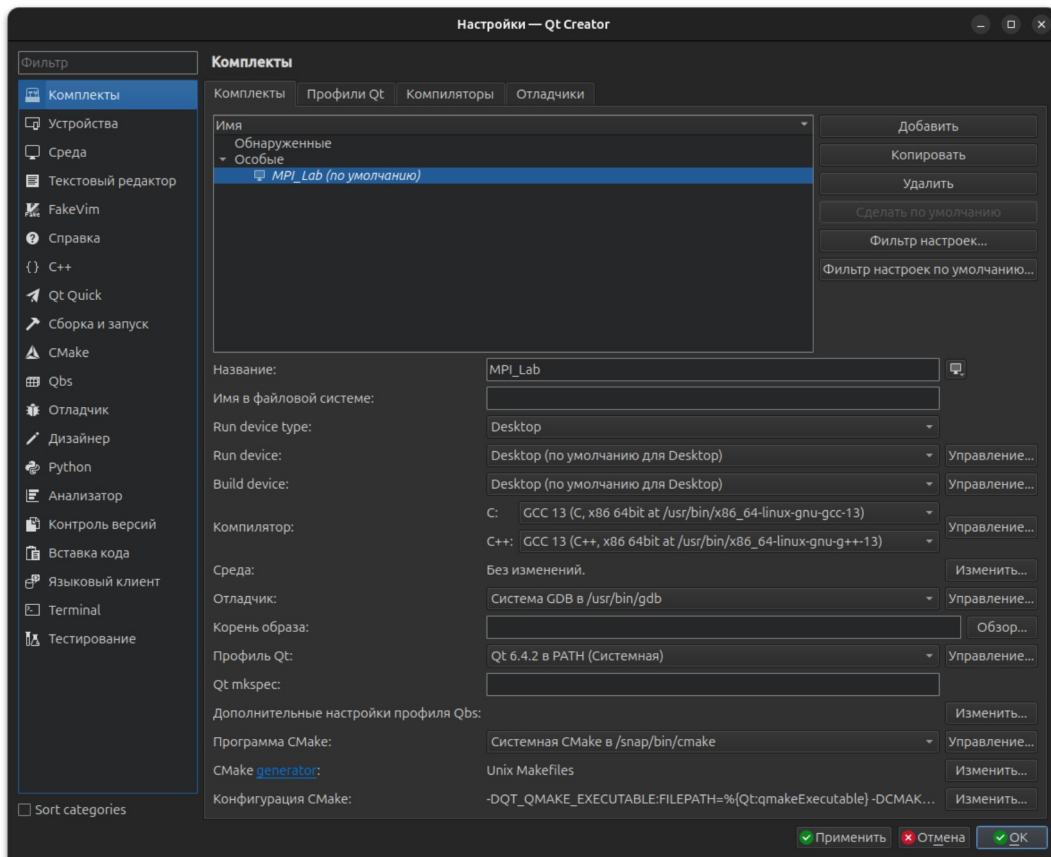


Рисунок 3. Настройка комплекта для запуска приложения

```
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

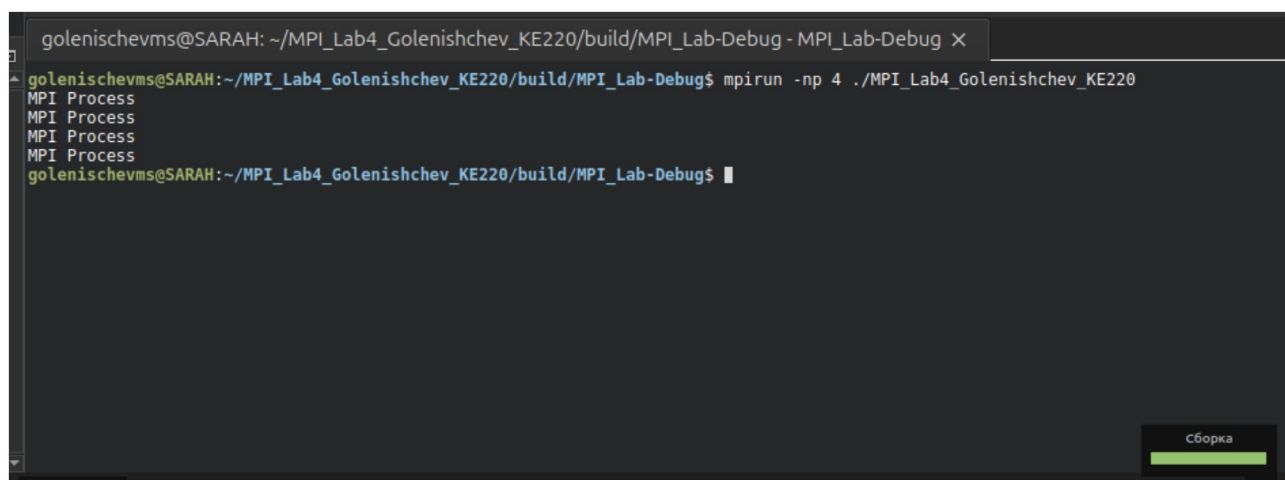


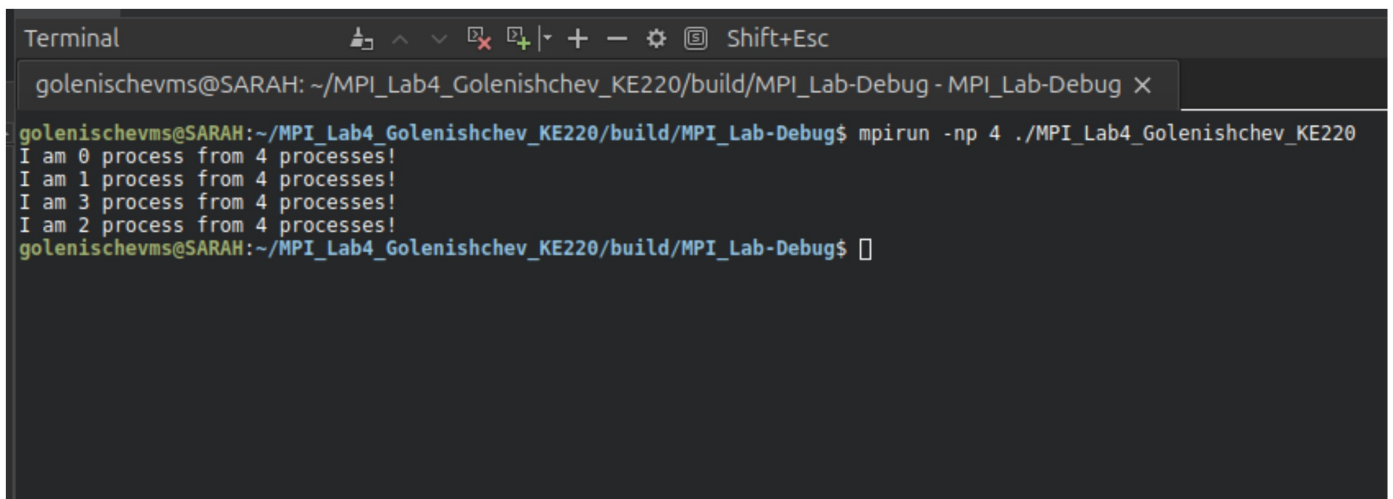
Рисунок 4. Запуск собранного исполняемого файла с MPI

### 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!»



```
Terminal
golenishevms@SARAH: ~/MPI_Lab4_Golenishchev_KE220/build/MPI_Lab-Debug - MPI_Lab-Debug X
golenishevms@SARAH:~/MPI_Lab4_Golenishchev_KE220/build/MPI_Lab-Debug$ mpirun -np 4 ./MPI_Lab4_Golenishchev_KE220
I am 0 process from 4 processes!
I am 1 process from 4 processes!
I am 3 process from 4 processes!
I am 2 process from 4 processes!
golenishevms@SARAH:~/MPI_Lab4_Golenishchev_KE220/build/MPI_Lab-Debug$
```

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>

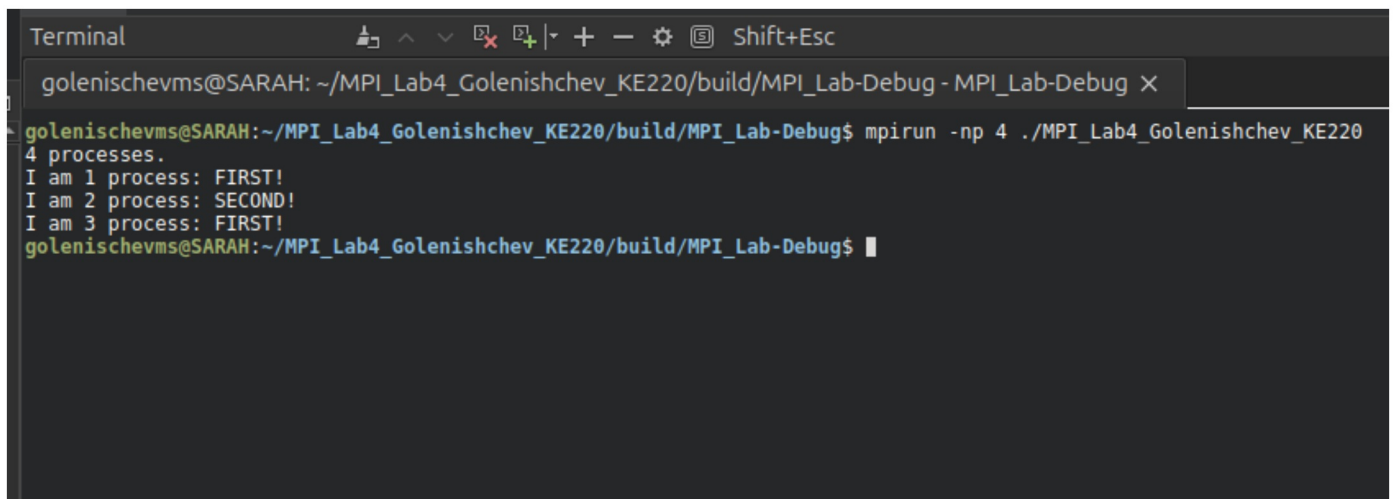
#include <stdio.h>
// Golenishchev Artem, KE-220 Task 12
int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    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



```
Terminal
golenishevms@SARAH: ~/MPI_Lab4_Golenishchev_KE220/build/MPI_Lab-Debug - MPI_Lab-Debug X
golenishevms@SARAH:~/MPI_Lab4_Golenishchev_KE220/build/MPI_Lab-Debug$ mpirun -np 4 ./MPI_Lab4_Golenishchev_KE220
4 processes.
I am 1 process: FIRST!
I am 2 process: SECOND!
I am 3 process: FIRST!
golenishevms@SARAH:~/MPI_Lab4_Golenishchev_KE220/build/MPI_Lab-Debug$
```

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.

2. 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()`.

3. Что такое коммунитор? Как учитываются входящие в него процессы?

Коммунитор в MPI — это группа процессов, которые могут взаимодействовать друг с другом. Главный коммунитор `MPI_COMM_WORLD` включает все процессы программы и используется по умолчанию. Входящие в коммунитор процессы определяются при его создании, а их количество можно узнать с помощью функции `MPI_Comm_size()`.

4. 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.

5. 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.