

МИНОБРНАУКИ РОССИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление 02.03.01 Математика и компьютерные науки

Отчет о выполнении

научно-исследовательской работы

**«Технологии параллельного программирования в
операционных системах Linux»**

Студент,

группа 5130201/20102

_____ Смирнов К.В.

Научный руководитель:

_____ Чуватов М.В.

«_____» _____ 2025г.

Санкт-Петербург, 2025

Реферат

Общий объем отчета: 40

Количество иллюстраций: 7

Количество использованных источников: 3

Количество приложений: 5

ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ, LINUX СИСТЕМЫ, ТЕХНОЛОГИЯ OPENMP, ТЕХНОЛОГИЯ OPENMPI, АЛГОРИТМ СЖАТИЯ LZ77

Объектом исследования являются технологии параллельного программирования в ОС на базе ядра Linux.

Цель работы - ознакомление с виртуальной средой VirtualBox, изучение настройки сетей и возможностей работы с ОС Linux, исследование технологий параллельного программирования OpenMP и OpenMPI в системах Linux и применение этих технологий на лабораторной работе по теории графов.

В процессе работы были установлены виртуальная среда VirtualBox и операционная система Fedora. Проведена базовая настройка сети, изучены основные возможности взаимодействия с операционной системой Linux, а также технологии OpenMP и OpenMPI. Подготовлена среда разработки и выполнено перемещение приложения лабораторной работы по теории графов.

В результате исследования были разработаны две программы на языке C++, использующие технологии OpenMP и OpenMPI для параллельного программирования. Программа, использующая технологию OpenMP, реализует параллельные вычисления на одном узле с помощью многопоточности. Приложение с OpenMPI спроектирована так, чтобы могла выполняться на 4 вычислительных узлах, включая узел на котором происходит запуск.

Содержание

Термины и определения	5
Перечень сокращений и обозначений	6
Введение	8
1 Основная часть	9
1.1 Установка и настройка среды виртуализации	9
1.1.1 Настройка разрешения имен	10
1.1.2 Настройка подключения по SSH	11
1.1.3 Настройка буфера обмена с внешней системой	12
1.2 Установка ПО для OpenMP. Тестовая программа.	13
1.3 Установка ПО для OpenMPI. Тестовая программа.	15
1.4 Директивы OpenMP, использованные в работе	18
1.5 Функции OpenMPI, использованные в работе	18
2 Реализация	23
2.1 Генерация случайного файла	23
2.2 Сравнение файлов	23
2.3 Коэффициент сжатия	24
2.4 Алгоритм LZ77	24
2.4.1 Класс LZ77	24
2.4.2 Метод <code>encode</code>	25
2.4.3 Метод <code>decode</code>	25
2.4.4 Сериализация	25
2.4.5 Параллельная реализация на MPI	25
3 Результаты работы	27
Заключение	29
Список использованной литературы	30

Приложение А. Исходный код lab6.cpp	31
Приложение В1. Исходный код General.h	32
Приложение В2. Исходный код General.cpp	32
Приложение С1. Исходный код LZ77.h	36
Приложение С2. Исходный код LZ77.cpp	37

Термины и определения

В настоящем отчете о НИР применяют следующие термины с соответствующими определениями:

Виртуальная машина - это программная среда, которая имитирует физический компьютер, позволяя запускать на нем операционную систему или несколько операционных систем одновременно на одном физическом компьютере

Операционная система — это программное обеспечение, управляющее аппаратными ресурсами компьютера и предоставляющее интерфейс для взаимодействия с пользователем и другими программами. Операционная система отвечает за управление процессами, памятью, устройствами ввода-вывода и обеспечивает безопасность, стабильность и эффективность работы компьютера

Сеть - это совокупность компьютеров и других устройств, которые связаны между собой для обмена данными и ресурсами

Fedora – это свободный и открытый Linux-дистрибутив, который служит тестовой площадкой и источником новых технологий для Red Hat Enterprise Linux (RHEL).

DNS - это система, которая преобразует доменные имена в IP-адреса и обратно

IP-адрес - это уникальный идентификатор компьютера или сетевого устройства в сети, используемый для маршрутизации пакетов данных

Nat-сеть - это технология, которая позволяет переводить IP- адреса между локальной сетью и внешней сетью. В случае использования Nat-сети, IP-адреса в локальной сети назначаются приватными адресами, а маршрутизатор выполняет преобразование адресов при отправке и получении пакетов данных между локальной и внешней сетью.

OpenMP - это набор директив компилятора, библиотек и средств программирования для параллельного программирования на общедоступных многоядерных системах. OpenMP позволяет разработчикам использовать потоки для распараллеливания кода и улучшения производитель-

ности. Он предоставляет простой и удобный способ добавления параллельности в программы на языках программирования, таких как C, C++ и Fortran

OpenMPI - это библиотека и стандарт для параллельных вычислений на распределенных системах. Она предоставляет набор функций и инструментов для обмена сообщениями и координации вычислительных задач между процессами, работающими на разных узлах в кластере или распределенной сети. OpenMPI позволяет эффективно использовать мощности множества компьютеров для параллельных вычислений

SSH - это протокол сетевой безопасности, который обеспечивает защищенное удаленное подключение и обмен данными между компьютерами. SSH используется для удаленного управления компьютерами, передачи файлов и выполнения команд на удаленных машинах. Протокол SSH обеспечивает шифрование данных и аутентификацию, обеспечивая безопасность при удаленном доступе к системам

SSH-ключ - это файл, содержащий криптографические ключи, используемые для аутентификации пользователя на удаленном сервере по протоколу SSH

VirtualBox — это программное обеспечение для виртуализации, которое позволяет запускать на одном компьютере несколько разных операционных систем одновременно в так называемых виртуальных машинах

Перечень сокращений и обозначений

В настоящем отчете о НИР применяют следующие сокращения и обозначения:

ВМ - виртуальная машина

ОС - операционная система

ПО - программное обеспечение

DNS - Domain Name System

IP - Internet Protocol

MP - Multiprocessing

MPI - Message Passing Interface

NAT - Network Address Translation

SSH - Secure Shell

Введение

Одной из самых популярных операционных систем является ОС на базе ядра Linux. Системы Linux актуальны благодаря своей открытости, безопасности и стабильности. Они широко используются в серверных решениях, облачных технологиях, встраиваемых системах и суперкомпьютерах. В данной работе использовался дистрибутив Fedora.

В рамках научно-исследовательской работы была поставлена задача изучить технологии параллельного программирования в операционной системе на базе ядра Linux. Необходимо было освоить основные принципы и методы параллельного программирования, а также понять особенности и преимущества использования этой ОС для выполнения параллельных задач.

Для разработки приложений с использованием технологий параллельного программирования на базе ОС Linux использовалось такое средство визуализации операционных систем как VirtualBox. VirtualBox — наиболее популярная программа виртуализации, которая имеет русифицированный интерфейс. С помощью нее можно создать виртуальные ОС Windows, Linux, macOS и Android.

Вариант дистрибутива: Fedora

Вариант сети: 172.16.110.0/24

Выполнение работы можно разбить на следующие этапы:

- установка и настройка VirtualBox, создание четырёх виртуальных машин;
- настройка локальной сети и подключения по SSH между машинами;
- установка и настройка ПО для работы с MP и MPI на виртуальных машинах;
- портирование программы из лабораторной работы из теории графов;
- запуск программ с использованием OpenMP и OpenMPI на VM.

1 Основная часть

1.1 Установка и настройка среды виртуализации

- **Установка VirtualBox**

Для создания приложений с применением технологий параллельного программирования на ОС Linux использовалась программа виртуализации операционной системы VirtualBox. Для эмуляции работы в операционной системе Linux необходимо скачать Fedora.

- **Настройка сети**

Для настройки сети системы необходимо перейти в панель «Инструменты», выбрать «Сеть», затем «Сети NAT» и нажать «Создать». При создании сети IPv4 префикс необходимо указать в соответствии с вариантом (172.16.110.0/24).

- **Создание виртуальной машины**

1. Для создания ВМ необходимо в меню VirtualBox выбрать «Создать». В появившемся окне задать имя ВМ, куда она будет установлена, а также выбрать заранее скачанный образ Fedora. После выбора оставшиеся поля заполнятся автоматически.
2. Выделение ресурсов осуществляется следующим образом: объем оперативной памяти устанавливается 2048 МБ, количество процессоров - 1, размер виртуального жесткого диска - 15 ГБ.
3. В настройках создаваемых виртуальных машин на вкладке «Сеть» необходимо выбрать для первого сетевого адаптера тип подключения «Сеть NAT», а в качестве имени выбрать ранее созданную сеть. Остальные сетевые адаптеры на вкладке «Сеть» необходимо выключить.

- **Установка операционной системы на ВМ**

После создания виртуальной машины ее необходимо запустить и установить ОС. Большинство параметров настройки ОС интуитивно понятны (базовые), оставшиеся настраиваем следующим образом:

1. В связи с отключением DHCP сеть необходимо настроить вручную. В качестве IP-адреса машины выбирается адрес

из доступного диапазона адресов(определяется вариантом - 172.16.110.0/24). Важно помнить, что первый адрес узла всегда будет занят виртуальным маршрутизатором VirtualBox, и он же является адресом шлюза по-умолчанию и адресом сервера DNS. Второй адрес также не рекомендуется использовать в качестве IP ВМ, так как могут возникнуть проблемы с подключением извне. Таким образом, назначаем первой ВМ адрес 172.16.110.101, второй - 172.16.110.102 , третьей - 172.16.110.103, четвёртой - 172.16.110.104.

2. Выбираем имя компьютера(например, node1..4) и домен(например, hpc). Имя домена для всех узлов в виртуальной сети должно быть одинаковым, а имена узлов — уникальными.
3. Далее необходимо провести настройку учетных записей суперпользователя(root) и пользователя. Учётная запись суперпользователя будет создана автоматически и для неё необходимо будет задать пароль. Пароль для суперпользователя задавать необязательно, тогда для выполнения действий, требующих привилегий, необходимо использовать команду sudo, если же пароль задан, то вход в систему от имени «администратора» или переключение в его контекст осуществляется с помощью команды su -. Учётную запись обычного пользователя можно назвать по своему усмотрению и для неё тоже необходимо будет задать пароль. Для простоты имя пользователя на всех машинах можно задать одинаковым.
4. Далее следует выбор дополнительного ПО. Здесь в качестве экономии дискового пространства рекомендуется отказаться от графического окружения и прочего и оставить только стандартные системные утилиты и SSH-сервер.

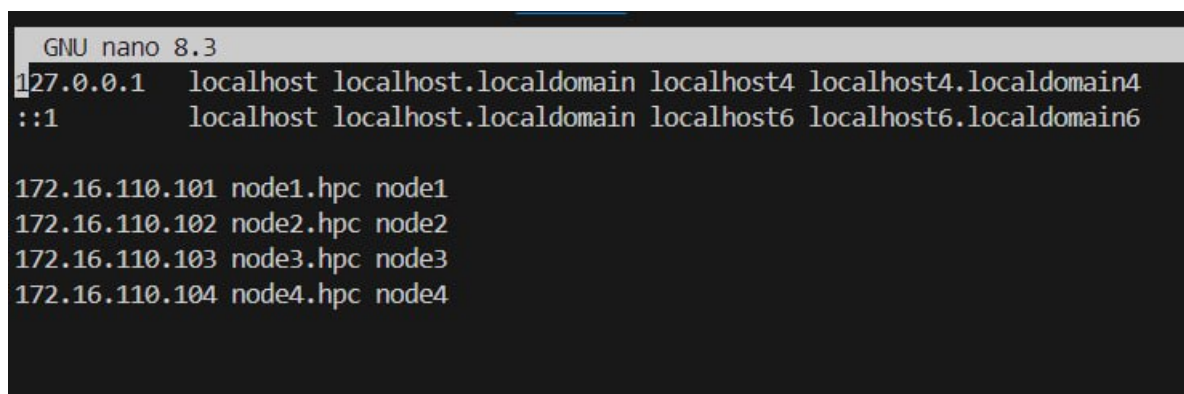
После всех шагов у нас будет виртуальная машина с установленной ОС и настроенной сетью.

1.1.1 Настройка разрешения имен

Для обращения к другим узлам виртуальной сети по их именам, а не IP адресам, необходимо настроить разрешение имён. Сделать это можно перечислением имен и сопоставлением им адресов в файле /etc/hosts, который позволяет разрешать имена узлов без настройки службы DNS. После успешной настройки (на каждом узле в файле hosts должны быть

перечислены все остальные узлы) получится обратиться к другому узлу командой `ping` уже с указанием имени узла, а не его IP-адреса.

В Linux файл `hosts` находится в папке `/etc/hosts`. Для редактирования `hosts` нужен доступ суперпользователя. Чтобы отредактировать файл `hosts`, необходимо ввести в терминал Linux команду `hosts: sudo nano /etc/hosts` (в контексте суперпользователя `nano /etc/hosts`) [1](#)



```
GNU nano 8.3
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

172.16.110.101 node1.hpc node1
172.16.110.102 node2.hpc node2
172.16.110.103 node3.hpc node3
172.16.110.104 node4.hpc node4
```

Рис. 1. Файл `/etc/hosts` с соответствием имен узлов

1.1.2 Настройка подключения по SSH

SSH (Secure Shell) - это протокол для безопасного удаленного доступа к другим компьютерам в сети.

При настройке ОС в качестве дополнительного ПО SSH-сервер уже был установлен на каждой из виртуальных машин. Статус службы можно проверить командой `sudo systemctl status ssh`. Служба должна быть запущена(`active(running)`), но если это не так, то ее нужно запустить командой `sudo systemctl start ssh`.

При стандартной авторизации по SSH пароль передается в незашифрованном виде по сети, что может быть опасным для безопасности. Вместо этого, для авторизации можно использовать ключи. Ключи - это пара файлов: закрытый ключ (`private key`) и открытый ключ (`public key`), которые используются для проверки подлинности пользователя.

Для генерации ключевой пары на каждом узле необходимо выполнить: `ssh-keygen`.

Открытый ключ должен быть размещен на удаленных узлах, к которым

нужно подключиться без ввода пароля. Для этого можно воспользоваться командой **ssh-copy-id user@remotehost**, где user - имя пользователя на удаленном хосте, а remotehost - адрес удаленного хоста, на который необходимо подключиться. Если имена пользователей одинаковые, как в данной работе, можно указывать только адрес или доменное имя указанное ранее в файле hosts (**ssh-copy-id remotehost**).

ssh-copy-id node2
ssh-copy-id skv@node2

После успешного выполнения команд при попытке подключения к удаленному хосту с помощью SSH не нужно будет вводить пароль. Чтобы проверить настройку ssh, можно подключиться с одной машины на другую с помощью команды **ssh remotehost**. Если удалось подключиться без ввода пароля, то все настроено верно. 2

```
skv@node1:~$ ssh node2
Web console: https://node2.hpc:9090/ or https://172.16.110.102:9090/

Last login: Sat Jun 28 18:36:31 2025 from 172.16.110.101
skv@node2:~$ ssh node3
Web console: https://node3.hpc:9090/ or https://172.16.110.103:9090/

Last login: Sat Jun 28 18:38:02 2025 from 172.16.110.102
skv@node3:~$ ssh node1
Web console: https://node1.hpc:9090/

Last login: Sun Jun 29 22:20:33 2025
skv@node1:~$
```

Рис. 2. Подключение по SSH

1.1.3 Настройка буфера обмена с внешней системой

Для того, чтобы была возможность пересылать файлы из основной ОС на виртуальные машины, необходимо настроить проброс портов сети NAT. Для проброса портов необходимо перейти в панель «Инструменты», выбрать «Сеть», затем «Сети NAT». В качестве адреса хоста нужно указать 127.0.0.1(localhost) - стандартный локальный адрес. Если подключиться извне не получится, то localhost необходимо поменять на адрес сетевого адаптера VirtualBox(можно найти в параметрах основной системы). Порт хоста необходимо указать больше 4096, так как предыдущие порты являются привилегированными. Адрес гостя соответствует IP адресу виртуальной машины, а порт гостя по умолчанию 22. 3

Основные опции		Проброс портов				
IPv4		IPv6				
Имя	Протокол	Адрес хоста	Порт хоста	Адрес гостя	Порт гостя	
node1	TCP	127.0.0.1	40201	172.16.110.101	22	
node2	TCP	127.0.0.1	40202	172.16.110.102	22	
node3	TCP	127.0.0.1	40203	172.16.110.103	22	

Рис. 3. Проброс портов

Для отправки файлов с основной ОС на виртуальную можно воспользоваться программой WinSCP.

1.2 Установка ПО для OpenMP. Тестовая программа.

- **Определение OpenMP**

OpenMP (Open Multi-Processing) — это набор директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью. Официально поддерживается Си, C++ и Фортран, однако можно найти реализации для некоторых других языков, например Паскаль и Java.

- **Принцип работы OpenMP**

В стандарт OpenMP входят спецификации набора директив компилятора, вспомогательных функций и переменных среды. OpenMP реализует параллельные вычисления с помощью многопоточности, в которой «главный» (master) поток создает набор «подчиненных» (slave) потоков, и задача распределяется между ними.

- **Основные возможности OpenMP:**

1. Директивы компилятора: OpenMP предоставляет специальные директивы, которые встраиваются в исходный код программы и указывают компилятору, какие участки кода должны выполняться параллельно. Например, директива `#pragma omp parallel` создает команду для запуска параллельной секции кода.
2. Работа с потоками: OpenMP позволяет создавать потоки выполнения (threads) для распараллеливания работы. Потоки могут выполняться параллельно на многоядерном процессоре, ускоряя выполнение программы.

3. Управление потоками: OpenMP предоставляет средства для управления потоками, такие как создание потоков, определение количества потоков, ограничение области видимости потоков и другие.
4. Синхронизация: OpenMP предоставляет механизмы синхронизации потоков, такие как директивы `#pragma omp barrier` для ожидания завершения работы всех потоков или `#pragma omp critical` для создания критической секции.
5. Распределение работы: OpenMP позволяет легко распределять работу между потоками с помощью директив, таких как `#pragma omp for`, которая автоматически разбивает цикл на части и распределяет их между потоками.

В Fedora используется пакетный менеджер `dnf`. Компиляторы `gcc` и `g++` для языков C и C++ соответственно, устанавливаются с помощью команды `sudo dnf install gcc gcc-c++`. Перед установкой рекомендуется обновить уже установленные пакеты с помощью команды `dnf upgrade`.

Поддержка OpenMP встроена в компилятор `g++` по умолчанию. Пример тестовой программы с использованием OpenMP представлен в листинге 1. При компиляции такой программы необходимо указать дополнительный флаг `-fopenmp`. Количество потоков можно указать с помощью переменной окружения `OMP_NUM_THREADS`. Пример команд для компиляции и запуска тестового файла:

```
export OMP_NUM_THREADS=4
g++ -fopenmp -o hello hello.cpp
./hello
```

Листинг 1. Минимальная программа для проверки работоспособности OpenMP.

```
1 #include <iostream>
2 #include <omp.h>
3
4 int main() {
5     #pragma omp parallel
6     {
7         std::cout << "Hello,_World_from_thread_" << omp_get_thread_num() << std::endl;
8     }
9     return 0;
10 }
```

1.3 Установка ПО для OpenMPI. Тестовая программа.

- **Определение OpenMPI**

OpenMPI (Open Message Passing Interface) - это библиотека, предназначенная для разработки параллельных приложений, которая обеспечивает возможность обмена сообщениями между процессами, работающими на различных узлах вычислительного кластера.

- **Принцип работы OpenMPI**

MPI-программа - это множество параллельных взаимодействующих процессов, которые работают каждый в своей выделенной области памяти. По сути это N независимых программ, которые общаются между собой в ходе работы.

- **Основные возможности OpenMPI:**

1. Механизмы передачи сообщений: OpenMPI предоставляет набор функций для отправки и приема сообщений между процессами, работающими на различных узлах кластера.
2. Управление процессами: OpenMPI позволяет создавать, управлять и завершать параллельные процессы, а также устанавливать связи между ними.
3. Поддержка различных архитектур: OpenMPI поддерживает работу на различных вычислительных архитектурах, включая кластеры с общей памятью (SMP), распределенные кластеры и другие.
4. Гибкость конфигурации: OpenMPI обеспечивает гибкую настройку для различных типов сетей, протоколов передачи данных и других параметров, что делает его универсальным инструментом для разработки параллельных приложений.

В качестве реализации MPI была выбрана библиотека **OpenMPI**. Она была использована благодаря своей широкой поддержке и совместимости с используемыми версиями ОС и инструментов разработки. На всех машинах OpenMPI была установлена с помощью команды

```
sudo dnf install openmpi openmpi-devel
```

После установки необходимо добавить путь к бинарникам OpenMPI в переменную окружения PATH. Для этого в файл `~/.bashrc` была дописана строка

```
echo 'export PATH=/usr/lib64/openmpi/bin:$PATH' >> ~/.bashrc
```

Эта команда добавляет путь к OpenMPI в PATH, чтобы не указывать его вручную при каждом запуске терминала.

После установки можно проверить работоспособность MPI, запустив простой тест из листинга 2. Компиляция и запуск программы выполняются с помощью команд:

```
mpicxx -o hello hello.cpp
mpirun -n 2 ./hello
```

Листинг 2. Минимальная программа для проверки работоспособности MPI.

```
1 #include <mpi.h>
2 #include <iostream>
3
4 int main(int argc, char** argv) {
5     MPI_Init(&argc, &argv);
6
7     int world_size;
8     MPI_Comm_size(MPI_COMM_WORLD, &world_size);
9
10    int world_rank;
11    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
12
13    std::cout << "Hello, world from rank_" << world_rank
14              << "_out of_" << world_size << "_processors" << std::endl;
15
16    MPI_Finalize();
17    return 0;
18 }
```

Перед запуском программы на нескольких виртуальных машинах необходимо убедиться, что на всех них установлены OpenMPI и компилятор g++. Чтобы упростить обмен скомпилированным исполняемым файлом и данными между машинами, была организована общая папка, доступная на всех узлах кластера.

Для этого на одной из машин (например, на главной или серверной) была создана директория:

```
sudo mkdir -p /mnt/shared
```


Для обеспечения полного доступа к этой папке всем пользователям на всех машинах были выставлены права:

```
sudo chmod 777 /mnt/shared
```

Далее настроен экспорт этой папки по протоколу NFS (Network File System). Для этого в конфигурационный файл NFS-сервера `/etc/exports` была добавлена строка:

```
/mnt/shared    172.16.110.0/24(rw,sync,no_root_squash,no_subtree_check)
```

Здесь `172.16.110.0/24` указывает, что доступ разрешён всем клиентам из подсети `172.16.110.0` с маской `255.255.255.0`, `rw` означает разрешение на чтение и запись, а остальные опции обеспечивают корректную работу с правами.

После изменения конфигурации нужно перезапустить сервис NFS:

```
sudo exportfs -a
sudo systemctl restart nfs-server
```

На всех остальных машинах-клиентах, где будут запускаться MPI-программы, общая папка была смонтирована в ту же директорию `/mnt/shared` с помощью команды:

```
sudo mount <IP_сервера>:/mnt/shared /mnt/shared
```

Для автоматического монтирования при загрузке в файл `/etc/fstab` на клиентских машинах добавляется строка:

```
<IP_сервера>:/mnt/shared    /mnt/shared    nfs    defaults    0 0
```

В итоге на всех виртуальных машинах появляется доступ к общей папке `/mnt/shared`, куда можно копировать скомпилированные файлы и данные, обеспечивая единое пространство для запуска и обмена.

Затем можно запустить программу с помощью команды:

```
mpirun -host node1,node2,node3,node4 /mnt/shared/hello
```

1.4 Директивы OpenMP, использованные в работе

- **#pragma omp parallel** - это директива препроцессора для языка программирования C/C++, которая используется для создания параллельных участков кода с использованием OpenMP.
- **#pragma omp for** - это директива OpenMP для языков программирования C/C++, которая используется для параллельного выполнения циклов. Эта директива позволяет разделить выполнение итераций цикла между несколькими потоками.
- **#pragma omp parallel for reduction(+:local_price)** - это директива OpenMP для языков программирования C/C++, которая используется для распараллеливания цикла с одновременным выполнением операции reduction над общей переменной local_price.
- **#pragma omp parallel num_threads(numThreads)** — директива, аналогичная **#pragma omp parallel**, но с явным указанием количества потоков, которое задаётся переменной numThreads. Это позволяет вручную контролировать степень параллелизма, например, в зависимости от количества доступных ресурсов или особенностей задачи.

1.5 Функции OpenMPI, использованные в работе

- **int MPI_Init(int *argc, char ***argv)**

Функция используется для инициализации среды выполнения параллельной программы.

Аргументы: Указатели на аргументы командной строки argc и argv, инициализируемые для использования в MPI.

- **int MPI_Finalize(void)**

Завершает работу MPI и освобождает все связанные ресурсы. После вызова другие функции MPI использовать нельзя.

- **int MPI_Comm_size(MPI_Comm comm, int *size)**

Получает общее количество процессов в указанном коммуникаторе.

– *MPI_Comm comm*: коммуникатор.

- *int* size*: указатель на переменную, в которую будет записано количество процессов.

- **int MPI_Comm_rank(MPI_Comm comm, int *rank)**

Определяет ранг (идентификатор) текущего процесса в коммуникаторе.

- *MPI_Comm comm*: коммуникатор.
- *int* rank*: указатель на переменную для хранения ранга текущего процесса.

- **int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm)**

Отправляет сообщение указанному процессу.

- *void* buf*: указатель на буфер с отправляемыми данными.
- *int count*: количество элементов.
- *MPI_Datatype datatype*: тип данных (например, MPI_INT).
- *int dest*: ранг процесса-получателя.
- *int msgtag*: тег сообщения.
- *MPI_Comm comm*: коммуникатор.

- **int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status* status)**

Принимает сообщение от указанного процесса.

- *void* buf*: буфер для приёма данных.
- *int count*: количество элементов.
- *MPI_Datatype datatype*: тип данных.
- *int source*: ранг процесса-отправителя.
- *int tag*: тег сообщения.
- *MPI_Comm comm*: коммуникатор.
- *MPI_Status* status*: структура для хранения информации о сообщении.

- **int MPI_Gatherv(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, const int recvcounts[], const int displs[], MPI_Datatype recvtype, int root, MPI_Comm comm)**

Собирает переменное количество данных от всех процессов на один процесс (root).

- *const void* sendbuf*: буфер с отправляемыми данными.
- *int sendcount*: количество элементов от каждого процесса.
- *MPI_Datatype sendtype*: тип отправляемых данных.
- *void* recvbuf*: буфер приёма (на процессе root).
- *const int recvcounts[]*: массив количества данных от каждого процесса.
- *const int displs[]*: массив смещений для каждого процесса.
- *MPI_Datatype recvtype*: тип принимаемых данных.
- *int root*: ранг процесса-получателя.
- *MPI_Comm comm*: коммуникатор.

- **int MPI_Scatterv(const void *sendbuf, const int sendcounts[], const int displs[], MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)**

Рассылает переменное количество данных от одного процесса ко всем остальным.

- *const void* sendbuf*: буфер с данными (только на процессе root).
- *const int sendcounts[]*: массив с количеством элементов для каждого процесса.
- *const int displs[]*: массив смещений в буфере отправки.
- *MPI_Datatype sendtype*: тип данных в отправке.
- *void* recvbuf*: буфер приёма.
- *int recvcount*: количество элементов, которые должен принять каждый процесс.
- *MPI_Datatype recvtype*: тип принимаемых данных.

- *int root*: ранг отправляющего процесса.
 - *MPI_Comm comm*: коммунитор.
- **int MPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)**
 Рассылает данные от одного процесса (*root*) ко всем остальным.
 - *void* buffer*: буфер с данными (и для отправки, и для приёма).
 - *int count*: количество элементов.
 - *MPI_Datatype datatype*: тип данных.
 - *int root*: ранг отправителя.
 - *MPI_Comm comm*: коммунитор.
 - **int MPI_Barrier(MPI_Comm comm)**
 Синхронизирует все процессы в коммуниторе: каждый процесс блокируется, пока все не достигнут барьера.
 - *MPI_Comm comm*: коммунитор.
 - **int MPI_Abort(MPI_Comm comm, int errorcode)**
 Немедленно завершает выполнение всех процессов в коммуниторе.
 - *MPI_Comm comm*: коммунитор, для которого завершается выполнение.
 - *int errorcode*: код ошибки, передаваемый при завершении.
 - **int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)**
 Выполняет операцию сокращения (суммирование, максимум и др.) и передаёт результат процессу *root*.
 - *const void *sendbuf*: отправляемый буфер.
 - *void *recvbuf*: буфер для результата (только на *root*).
 - *int count*: количество элементов.
 - *MPI_Datatype datatype*: тип данных.
 - *MPI_Op op*: операция (например, *MPI_SUM*).

- *int root*: ранг процесса-получателя.
- *MPI_Comm comm*: коммунитор.

2 Реализация

2.1 Генерация случайного файла

Функция `generateFile` предназначена для генерации случайного файла и принимает следующие аргументы:

1. `const std::string &filename` — имя выходного файла;
2. `const std::string &alphabet` — строка, задающая алфавит допустимых символов;
3. `int total_size` — общее количество символов для генерации;
4. `int rank` — ранг текущего MPI-процесса;
5. `int size` — общее количество процессов MPI.

Нулевой процесс (`rank = 0`) считается корневым и выполняет:

- разбиение общей задачи на части;
- распределение задач между остальными процессами;
- сбор результатов генерации.

Передача и сбор данных реализуются с помощью `MPI_Gather`. Для локальной генерации каждого фрагмента используется вспомогательная функция `generateChunk`, в которой применяется директива `OpenMP`:

```
#pragma omp parallel for
```

что позволяет распараллелить генерацию символов по потокам.

2.2 Сравнение файлов

Функция `areFilesEqual` предназначена для побайтового сравнения двух файлов. Её параметры:

1. `const std::string &filename1` — имя первого файла;
2. `const std::string &filename2` — имя второго файла;

3. `int rank` — ранг текущего MPI-процесса;
4. `int size` — общее количество процессов MPI.

Процесс с рангом 0:

- загружает оба файла;
- определяет размер и делит их на фрагменты;
- рассылает части другим процессам с помощью `MPI_Send`;
- использует `MPI_Bcast` для передачи размеров;
- собирает результат обратно через `MPI_Recv`.

Для распараллеливания сравнения используется:

```
#pragma omp parallel for reduction(&&:isEqual)
```

что обеспечивает логическое И между всеми результатами потоков.

2.3 Коэффициент сжатия

Функция `calculateCompressionRatio` вычисляет:

- размер исходного файла;
- размер сжатого файла;
- коэффициент сжатия:

$$CR = \frac{\text{исходный размер}}{\text{сжатый размер}}$$

Примечание: если входной файл пуст, выводится предупреждение.

2.4 Алгоритм LZ77

2.4.1 Класс LZ77

Класс содержит основные константы:

- `WINDOW_SIZE` = 4096 — размер окна поиска совпадений;
- `LOOKAHEAD_SIZE` = 18 — размер буфера предпросмотра.

2.4.2 Метод `encode`

Метод принимает строку `input` и возвращает вектор токенов `Token` вида `(offset, length, next_char)`:

- `offset` — смещение назад до совпадения;
- `length` — длина совпадения;
- `next_char` — следующий символ после совпадения.

Если совпадение не найдено, добавляется токен с `offset = 0, length = 0` и текущим символом.

2.4.3 Метод `decode`

Метод восстанавливает исходную строку по вектору токенов:

- копирует ранее декодированные символы согласно `offset` и `length`;
- добавляет `next_char`, если он не равен нулю.

2.4.4 Сериализация

Для записи токенов в файл и последующего чтения реализованы функции:

- `serializeTokens` — преобразует вектор токенов в бинарный массив;
- `deserializeTokens` — восстанавливает токены из бинарного массива.

2.4.5 Параллельная реализация на MPI

Метод `encodeFileMPI`:

- Процесс 0 читает весь файл;
- Делит входные данные на части и рассылает их через `MPI_Scatterv`;
- Каждый процесс выполняет локальное сжатие;

- Результаты собираются через `MPI_Gatherv` и сохраняются в файл.

Метод `decodeFileMPI`:

- Декодирование выполняется только на процессе 0;
- Остальные процессы находятся в ожидании на `MPI_Barrier`;
- Причина — необходимость доступа к ранее декодированным данным при обработке токенов.

3 Результаты работы

В результатах работы (Рис. 4-7) представлены примеры работы программы.

```
skv@node1:/mnt/shared$ export OMP_NUM_THREADS=2
skv@node1:/mnt/shared$ mpirun --host node1,node2,node3,node4 /mnt/shared/lab0
[node1:04080] plm:ssh: Warning: setpgid(4084,4084) failed in parent with errno=Permission denied(13)
[INFO] Используется количество потоков: 2
Файл 'random_text.txt' успешно создан.
Кодирование и декодирование с использованием LZ77
Тест пройден для LZ77: декодированный файл совпадает с исходным.
Размер исходного файла: 10000 байт
Размер сжатого файла: 21558 байт
Коэффициент сжатия: 0.463865
skv@node1:/mnt/shared$ █
```

Рис. 4. Пример работы программы для 2 потоков

```
skv@node1:/mnt/shared$ mpirun --host node1,node2,node3,node4 /mnt/shared/lab0
[node1:03891] plm:ssh: Warning: setpgid(3894,3894) failed in parent with errno=Permission denied(13)
[INFO] Ограничено количество потоков до 4 (было 1000)
Файл 'random_text.txt' успешно создан.
Кодирование и декодирование с использованием LZ77
Тест пройден для LZ77: декодированный файл совпадает с исходным.
Размер исходного файла: 10000 байт
Размер сжатого файла: 21666 байт
Коэффициент сжатия: 0.461553
skv@node1:/mnt/shared$ █
```

Рис. 5. Пример работы программы для 100 потоков (ограничение)

```
2387/C5c0eCa\10a77ce57fa8BE42e66 E.FD79C\a7c1DC27470.B164abd\1f3043CF67c60 1EEA8.d0b337c d0edCD2dFEfbf\9CC97e11d7B2Be7 1fe2128d.Dc44AEb86d6
9d392dbE a4.EbD444e03 7086\11 219e2AD9 8 .4B... 5cfe\B.08E... Cfab2\61acfa 5E9C4Fab1057Ab4679Eb88B 5d1bcfc8\46.8\7AA9DCE3\A fc6BEBd.84dAbc
e3F8Dc4fB2DCb\CF65ED\96ED0063 \58b.Bcfbed5E15\55492 d8 Bfe977 FEb06b8efe E4e85c.BCDBcfD\af99\B95923f4310402B eebb46daa79\B8ca4Bb8
efB2514A6G0C75E42D1e27a768aa3E6cD12803b 61AB9E8.06A67Efe641860fA7a3f832096f8acd bf51060\96fAGD2 2815dadaaC297BAe8f9C\A1c\54.b8d3BB.
1F469Ea02e0 b \.9.b5E\1C9C03258aafbf\c B54AEa.E726FADecbe05ed\71f62.5E.Dbb8da8Dd\5b\ d7B 7DAA3d7BCce39b1fccc..
Fbf\7CBf4cb9520A6Af\72d36f4482d079 3\JA185EBA\17 a6 c\6.8f97d6 7 95\cCEB.a\..7\ED5 77B47a31955bfb8a 2FE.2997C\1..bc\8Ecc3f89C cE.
AE93Eae3E03C2b0c40.3Cdf E5aE3de1dab..Ea61 AFCB1a4aG08Bf5\bb5eFCCD\f5A37ACf07C5A54642d2defcc f.E3fb 37d\8dD6C1e4bBAF\D64ffca4D119Df3a
59cc7fFcaB\CA4D6F 31E97ceF6dcac5A d86BD5FBC4826aFcf8be1A7\BddBFD673eE5 \cd0f4 FE4fD344 .CB9cecaEf8cf8b9f3 d34 36DcA90Bbd b9ff6f.6e125c4edAcEc.
33aef5e1e1C1E70aEDc4.fF66bcea.Aac0cC1a7a8aa E5bc8a\61\7C.4Bb6b1ee02 69D595\69.D d5C5E 6a6AC6D2Fb8FAFFb23.B 17F6c0B F9dd8 E.4\A8dbba 3\27A3..
EaB6da\eeFbccc4E7E8.F\D83B.E1d4eCdb4F61\dc493a4a.E94CcaEC06d9\deB8e 9AF9.Ebbcb6f.8748.DF3\A\bb10C9b88Ef0dd3\CABA5b9C26c\ec5B
0c656eBfAe0Dca\0aacb6e0BE49aa15\D\CBdfdbage1d E2\0.fdfDb499ccf5EfaD4eb05AE83\9E0c c.2D7C97F01\583df7f72faa5eAf2d.848D\EA46\478Dda8c5EFD45C.
419A4eA.af4B8 cdbb182Dfd627c650ac0BfCFa0B8.ea77993 5b0c\De48.eE\F05945A1D0C 11\BEAB bac2FEdbeedB46F52Ef .B0\9Dc5B8ac4aD84
e0aD39Af99fae7E2c\6194De\8943C4 9b2Ae5dC552 bf\aadE\4\362f8 F7fdF1b1\c1Fc.4a8fC1Fa266ECac34aa\9AfD8.8fA.2b8540d.\6acac9E d\51bfb6b605.9a64.
2aa5C42D2c.6e5d6c.B8051A9fB4E9CBF6EF.A\AFbB \.0cb77c71Fbe549187\5f\df30ff5A72a4F7be 7Ac56ce 3b3177F6aa1C0 .ce51D0c1dc\ 97.DaEba8d\Ec43b6.d73E3BF3D
FFd00.47CD A33c102Bccf439 dffBACCe154eb3e658CAce3Cdb82a1F2.7 FA8AF85f2d5Bdd7a0E30515A260Cc 751A8d02f18\E5\F88c86D1a47F\cf332CC3e15c7c\7.
C3f0fb11D189 c.Da\..B8787f38\1c11cACdE267Ce63642A75ba16aB8F.DE19AD421.a.a451AB27324E041ea18A87\DbBEbE4 f\D..aA 7bc8 221\A85
17bb2\ecceCD0E5\38ACD6Fc e.ef1B5aA\7cfBAA\277caF Ffe\c.570\..8cb7d\ 208C62Dc.c1f5\1 65cc89AD5a1dfb549e1fDbd2.F0c065.5e452f6f BfcFe\D288C2fcc\
351dCfb1ea.FE01a4FC6e0\97413CBb47.b2f32a\B.5d0\6.fed3d85e3B8D6e0f3dFA49.38be74EEa1aa852A47b1 dFaE.4F6351 e2beaf72A161165 2. FA3675630EBA67b
F6e43B7089C 6308edd5 a7b56EB6ee\ 5FA.CD89 0Ca26eBfc83bA8f0cfcb870d1C.F\1bb8D2.BadcdF4e DAF e9ddED078db.09C\cfECa2C80b 924E\Aabdd.D80A E7.9e
50AA1CD.A1\AF615Ad5bc.abE\7952de0 \d0a9\2\04d 2801.b7cfDeE8002cc2\22926Fd.90D99d40 72bf\c9\6e0dc9bc6cBE.9f6 .ce57a8f78d30Ae48B85e0da2cb538E8e2.
e1bc3.8EF9ce eE3a\6fe0.DC\B85\A20845 9ed42Ce5f..3 ca\ae50D56adD47ca 75735B9f464bfa DBd0cb3 \.d8d.56\ea273c5D0abAC \4AD6c86\c83c66C7d79bcfF8.
07D4c1b01\ec7 \51..3BE8b\8a5fF8e3bd0a67de688 50\751FcCFd9A1C5f2 888D\fb3e3Bac4CE6 c\cdC5d3dAbc4EF8.eC. 816915fbB0eA2\ec5.1B\29
FF2e92d06D41432fed2e0bda2E 870fb EcCd03299fde66AAa9. DB80E03b3.B5E2C1\5 .4a\AFC.DC.E90\ee2000\3A20E54e41a707ffe\c6.ed9\aa86efC69ecFac673\CD0ba9A.7
\5b7A5c..A.c96 5A. BEF\Aab.9.fccDfCf\daA43257 d72C4EF8dc0.edA C2ebc182C 0a27Eaf9ACC2A15\B01B.D8Bf 69FbDeF1EA08\acBb6A1EF edFcdC\ba5decba
Cae123f8d759Eb955a E\c43\c9CEAF571d2.E75e1Ed.388d26.d94f908348b035fC7fAC2f3FF294D0F3CDaA6266Ecaf05e9a4a68daE.e2cb.BCC0d4986C7749
\3B16E7babB00d3A4B648dbdC3ECAe88c 0 ced10994caA6bFEff cBaC40897 D9 324 A9 FDD.73bc499adA24.f6d7043\18F.FF00Fcbad\B03. \CF\03aE69da\01aBE578A5d.
7f3CD587922C82e\8de9FE5F63da15\723CF5EEa7AE0CA f775A1..F 875Efaa03b61.9c4d7Ed0f11B\1e.d04CBEC6aB9cDe2fD85D0A8.e7ECbf1be39c9AB2e1
2f19cbAAE3\1cd4Fb0f17Dbdbcc66a2B1f0f59\262CBF86\EDA9ac020BACff\dBDBEEBAd.b6fda 4.43 715 f d0ed3C52 0 4e0
492DeAD0E8b9D05D41152FE3584Ce001E7255c99B25deacF60A\FFdd 83a17dFd.ED4E5059.ab7E196c4E01FC.BFA68Af.ec0ab 992D7F5 943Ddf1A cD8FAce5.0f Ed0.
e5a492A0b2B4EA8fEa30f43 acc34AC0dfb286f45f6\2Ee3.EbcAFBA 0437a22 5EC1dd.dbbf6f61ef.2faca8FE.4c7b \CeC1Ad7b32Ce2.Ed.F3F dF2e4E.B.
b15\7EfedA5679bB4A5E D aeBCA73.ABAf97c D6dF3Cb306d7b74B45Cea9E.1ECB34B851A1d\4baad9EcDc00C7 6ECf83cdcfE5F8526\..C1bdcD73.76443a.
082cfCB889d8c2fbEDD6ACfaB96d5f44A8aE\225c74.3fA44bef5B7448ecaAaAe1\B842C8aa\3228aDf03cF9ac429a 2EC\5.2db2ce.F1.FeF0c34f9F78CA83Ef0\4DD8902.
CC458.9fda8ad695cfD79962FEf36AA586f5b7d0G3A11b EbdA02339\c5b\daF6b75EC21\9f.79.5\B0366e.59321c4F1D8060739acEb7Bb7ad0\c98ce621DA00f.7e5 \da838523
E\bdac8d7b6c8ff0EC.D940eBC.8f 6AcE.95C392 2B807dadcEacF4.D8DfA028b23ef2e0fb..8DBA.50eD..eB1fD.Bce0c 0F7\337B\5Aa61020
FD67b0D280d2FA2DEaD33E59B3677C4CbcB8e84A E9. ade \.FA89 3E9f7e\5ED1d02\05\605\F78Eb1\3ab4fB88578F20daCeb67aFb1A4af7Bdb7bDEc03a79 .
FFC2e06fc39f73f68D8fcF0C CC0f3cd9a 7A2F8C94c95EC00B16f2\Cfd8AD4d8221c8B8874610 b 9.0E.C445354ced FDE94 21adC339b\42a\bd5bcdDcA6685aE43C.\0f
b1f1f.47E3C8F847B.04ab425b.BC4F8fdab5Aed9\B1d9D0\c0fB371F068E.3E.Ca8b\7Bff3EF3977943Bd06cAe03e7F9c\85.a360dFd\baD1.a2.3.3395990c8C81eA0CF6.3
F0b216fC36f\6\DFa805..A20560\cd4d50badA81fBbf61 eb8 a890B83.96ba.c9F288bbcfb5568da8796.6E2 a2CcEaDe.1b707FF30EE5\57D0\ef.
ce0df19AF5E8fde46AF4AD09CE8AB5A3ab0b\F\fb f874CAFfa5DaD4067\c dc.FbFAA280.f09cd61.2F\f1ECE4dedcf2.8a5A3EA200604.dC3e EB7f\ad25F5800dbcfA3.
DaD\B7E1fCD9b.16aEF3Ea2CCF7690.0\eb1.2C22DFEE774D1FED.D0\ 477ce484C72cf393d4C.ab6855AAAA899C83ce.a9 89C9baf0A0898cbfB\3cc1Ad98f5Deb591b1.
```

Рис. 6. Сгенерированный файл

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded Text
00000000 00 00 00 00 32 7F 00 00 00 00 33 8B 00 00 00 00 2 3
00000010 42 07 00 00 00 00 37 00 00 00 00 00 5C 00 00 00	B 7 \ . . .
00000020 00 00 43 00 00 00 00 00 35 01 00 00 00 00 63 00	. . C 5 c .
00000030 00 00 00 00 30 7F 00 00 00 00 45 00 03 00 01 00 0 E
00000040 41 00 08 00 01 00 31 00 06 00 01 00 61 00 07 00	A 1 a . . .
00000050 01 00 37 00 0D 00 01 00 65 00 0E 00 01 00 37 00	. . 7 e 7 .
00000060 00 00 00 00 66 00 08 00 01 00 38 00 17 00 01 00 f 8
00000070 45 00 00 00 00 00 34 00 1C 00 01 00 65 00 00 00	E 4 e . . .
00000080 00 00 36 00 01 00 01 00 20 00 07 00 01 00 2E 00	. . 6
00000090 00 00 00 00 46 00 00 00 00 00 44 00 10 00 01 00 F D
000000A0 39 00 15 00 01 00 5C 00 12 00 01 00 37 00 21 00	9 \ 7 ! .
000000B0 01 00 31 00 09 00 01 00 43 00 13 00 01 00 37 00	. . 1 C 7 .
000000C0 16 00 01 00 37 5C 25 00 01 00 2E 30 1C 00 01 00 7 \ % 0 . . .
000000D0 31 42 18 00 01 00 34 64 10 00 01 00 62 61 00 00	1 B 4 d b a .
000000E0 00 00 64 42 30 00 02 00 66 31 3E 00 01 00 30 42	. . d B 0 . . . f 1 > . . 0 B
000000F0 09 00 01 00 33 36 15 00 01 00 46 32 0E 00 01 00 3 6 F 2 . . .
00000100 37 66 1C 00 01 00 36 43 09 00 01 00 20 41 0E 00	7 f 6 C A . .
00000110 01 00 45 35 01 00 01 00 41 5C 37 00 01 00 2E 63	. . E 5 A \ 7 c
00000120 16 00 01 00 30 39 19 00 01 00 33 35 01 00 01 00 0 9 3 5 . . .
00000130 37 66 10 00 01 00 20 61 08 00 01 00 36 32 09 00	7 f a 6 2 . .
00000140 01 00 65 44 04 00 01 00 43 5C 32 00 01 00 32 61	. . e D C \ 2 . . . 2 a
00000150 04 00 01 00 46 46 16 00 01 00 66 31 12 00 01 00 F F f 1 . . .
00000160 66 2E 2B 00 01 00 39 20 0B 00 01 00 43 38 03 00	f . + 9 C 8 .
00000170 01 00 37 32 11 00 01 00 31 61 01 00 01 00 64 63	. . 7 2 1 a d c
00000180 05 00 01 00 42 41 13 00 01 00 42 34 08 00 01 00 B A B 4 . . .
00000190 37 35 2D 00 02 00 66 33 05 00 01 00 32 32 04 00	7 5 - f 3 2 2 .
000001A0 01 00 32 37 2F 00 01 00 64 46 30 00 01 00 44 44	. . 2 7 / d F 0 . . . D D
000001B0 2B 00 01 00 34 37 01 00 01 00 41 35 23 00 01 00	+ 4 7 A 5 # . . .
000001C0 62 30 0A 00 01 00 36 31 31 00 02 00 20 66 22 00	b 0 6 1 1 f " .
000001D0 01 00 64 5C 3A 00 01 00 39 36 31 00 02 00 62 46	. . d \ : 9 6 1 . . . b F
000001E0 0E 00 01 00 20 32 5D 00 01 00 34 35 18 00 01 00 2] 4 5 . . .
000001F0 45 2E 60 00 02 00 34 62 01 00 02 00 65 39 45 00	E . ` 4 b e 9 E .
00000200 01 00 33 5C 0D 00 01 00 20 38 72 00 02 00 38 31	. . 3 \ 8 r 8 1
00000210 1C 00 01 00 5C 43 01 00 02 00 20 43 31 00 02 00 \ C C 1 . . .
00000220 39 32 35 00 02 00 41 31 31 00 01 00 39 2E 09 00	9 2 5 A 1 1 9 . . .

Рис. 7. Закодированный файл

Заключение

В рамках выполнения научно-исследовательской работы были изучены основы работы с дистрибутивом Fedora ОС Linux, а также проведена установка и конфигурация среды виртуализации VirtualBox и базовые настройки сети. В процессе работы были созданы четыре виртуальные машины, между которыми были настроены сеть NAT и подключение по SSH. В ходе выполнения основной части работы были освоены базовые принципы технологий параллельного программирования OpenMP и OpenMPI. Было произведено портирование лабораторной работы по теории графов для запуска ее на нескольких процессах.

Список литературы

- [1] Учебник по OpenMP – Блог программиста // Блог программиста
URL: <https://pro-prof.com/archives/4335> (дата обращения: 27.06.2025).
- [2] А.С. Антонов Параллельное программирование с использованием технологии OpenMP. - Москва: МГУ, 2009 (дата обращения: 27.06.2025).
- [3] Часть 1. MPI — Введение и первая программа // Хабр URL:
<https://habr.com/ru/articles/548266/> (дата обращения: 27.06.2025).

Приложение А. Исходный код lab6.cpp

```
1  #include <mpi.h>
2  #include <omp.h>
3  #include <string>
4  #include <iostream>
5  #include "General.h"
6  #include "LZ77.h"
7
8  using namespace std;
9
10 int main(int argc, char *argv[])
11 {
12     MPI_Init(&argc, &argv);
13
14     int rank, size;
15     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
16     MPI_Comm_size(MPI_COMM_WORLD, &size);
17
18
19     int max_threads = omp_get_max_threads();
20     if (max_threads > 16) {
21         omp_set_num_threads(16);
22         if (rank == 0)
23             cout << "[INFO]_Ограничено_количество_потоков_до_4_было_" << max_threads << "
24             " << endl;
25     } else {
26         omp_set_num_threads(max_threads);
27         if (rank == 0)
28             cout << "[INFO]_Используется_количество_потоков:" << max_threads << endl;
29     }
30
31
32     setlocale(LC_ALL, "Russian");
33
34     string alphabet = "abcdefABCDEF_0123456789\\\\";
35     string inputFileName = "random_text.txt";
36     string encodedFileNameLZ = "encoded_text_lz77.bin";
37     string decodedFileNameLZ = "decoded_text_lz77.txt";
38     int total_size = 10000;
39
40     generateFile(inputFileName, alphabet, total_size, rank, size);
41
42     // LZ77
43     if (rank == 0)
44         cout << "Кодирование_и_декодирование_с_использованием_LZ77" << endl;
45
46     LZ77 lz;
47
48     lz.encodeFileMPI(inputFileName, encodedFileNameLZ, rank, size);
49     MPI_Barrier(MPI_COMM_WORLD);
50     lz.decodeFileMPI(encodedFileNameLZ, decodedFileNameLZ, rank, size);
51
52     // Проверка для LZ77
53     bool isEqual = areFilesEqual(inputFileName, decodedFileNameLZ, rank, size);
```

```

54
55     if (rank == 0){
56         if (isEqual)
57             cout << "Тест_пройден_для_LZ77:_декодированный_файл_совпадает_с_исходным." <<
             endl;
58         else
59             cout << "Тест_не_пройден_для_LZ77:_декодированный_файл_не_совпадает_с_исходным.
             " << endl;
60     }
61
62     // Вычисление коэффициента сжатия для LZ77
63     if (rank == 0)
64         calculateCompressionRatio(inputFileName, encodedFileNameLZ);
65
66
67     MPI_Finalize();
68     return 0;
69 }
70
71

```

Приложение В1. Исходный код General.h

```

1  #pragma once
2
3  #include <mpi.h>
4  #include <omp.h>
5  #include <fstream>
6  #include <iostream>
7  #include <random>
8  #include <string>
9  #include <sstream>
10
11 void generateFile(const std::string &filename, const std::string &alphabet, int total_size, int rank, int
    size);
12
13 bool areFilesEqual(const std::string &filename1, const std::string &filename2, int rank, int size);
14
15 void calculateCompressionRatio(const std::string &originalFile, const std::string &compressedFile);
16

```

Приложение В2. Исходный код General.cpp

```

1  #include "General.h"
2
3  using namespace std;
4
5  void generateChunk(std::string &chunk, const std::string &alphabet, int chunkSize) {
6      int alphabetSize = alphabet.size();

```



```

7     std::random_device rd;
8     std::mt19937 generator(rd());
9     std::uniform_int_distribution<> distribution(0, alphabetSize - 1);
10
11     chunk.resize(chunkSize);
12
13     #pragma omp parallel for
14     for (int i = 0; i < chunkSize; ++i) {
15         char randomChar = alphabet[distribution(generator)];
16         chunk[i] = randomChar;
17     }
18 }
19
20 void generateFile(const std::string &filename, const std::string &alphabet, int total_size, int rank, int
    size)
21 {
22     // Расчёт количества байтов на каждый процесс с учётом остатка
23     std::vector<int> counts(size), displs(size);
24     int base = total_size / size, rem = total_size % size;
25
26     for (int i = 0; i < size; ++i) {
27         counts[i] = base + (i < rem ? 1 : 0);
28         displs[i] = (i == 0 ? 0 : displs[i - 1] + counts[i - 1]);
29     }
30
31     // Генерация куска соответствующего размера
32     std::string chunk;
33     generateChunk(chunk, alphabet, counts[rank]);
34
35     // Сборка на rank 0
36     if (rank == 0) {
37         std::ofstream file(filename, std::ios::binary);
38         if (!file.is_open()) {
39             std::cerr << "Не_удалось_открыть_файл_для_записи.\n";
40             MPI_Abort(MPI_COMM_WORLD, 1);
41             return;
42         }
43
44         std::vector<char> global_data(total_size);
45         MPI_Gatherv(chunk.data(), counts[rank], MPI_CHAR,
46                     global_data.data(), counts.data(), displs.data(), MPI_CHAR, 0,
47                     MPI_COMM_WORLD);
48
49         file.write(global_data.data(), global_data.size());
50         file.close();
51         std::cout << "Файл_" << filename << "_успешно_создан.\n";
52     } else {
53         MPI_Gatherv(chunk.data(), counts[rank], MPI_CHAR,
54                     nullptr, nullptr, nullptr, MPI_CHAR, 0, MPI_COMM_WORLD);
55     }
56 }
57
58 bool areFilesEqual(const std::string &filename1, const std::string &filename2, int rank, int size)
59 {
60     std::ifstream file1, file2;

```

```

61     std::vector<char> buffer1, buffer2;
62     int fileSize1 = 0, fileSize2 = 0;
63     bool stopEverything = false;
64
65     // Чтение файлов и проверка размеров на корневом узле
66     if (rank == 0)
67     {
68         file1.open(filename1, std::ios::binary);
69         file2.open(filename2, std::ios::binary);
70
71         if (!file1.is_open() || !file2.is_open())
72         {
73             std::cout << "Ошибка_при_открытии_файлов_для_сравнения." << std::endl;
74             stopEverything = true;
75         }
76
77         if (!stopEverything)
78         {
79             file1.seekg(0, std::ios::end);
80             file2.seekg(0, std::ios::end);
81             fileSize1 = file1.tellg();
82             fileSize2 = file2.tellg();
83
84             if (fileSize1 != fileSize2)
85             {
86                 stopEverything = true;
87             }
88         }
89     }
90
91     MPI_Bcast(&stopEverything, 1, MPI_C_BOOL, 0, MPI_COMM_WORLD);
92     if (stopEverything)
93     {
94         return false;
95     }
96
97     // Сообщаем всем процессам размер файлов
98     MPI_Bcast(&fileSize1, 1, MPI_INT, 0, MPI_COMM_WORLD);
99
100    // Размеры чанков
101    int baseChunkSize = fileSize1 / size;
102    int remaining = fileSize1 % size;
103
104    // Размер чанка для текущего процесса
105    int start = rank * baseChunkSize;
106    int end = (rank == size - 1) ? (start + baseChunkSize + remaining) : (start + baseChunkSize);
107    int chunkSize = end - start;
108
109    buffer1.resize(chunkSize);
110    buffer2.resize(chunkSize);
111
112    // Корневой процесс распределяет чанки
113    if (rank == 0)
114    {
115        for (int i = 1; i < size; ++i)
116        {

```

```

117         int procStart = i * baseChunkSize;
118         int procEnd = (i == size - 1) ? (procStart + baseChunkSize + remaining) : (procStart +
baseChunkSize);
119         int procSize = procEnd - procStart;
120
121         std::vector<char> procBuffer1(procSize);
122         std::vector<char> procBuffer2(procSize);
123
124         file1.seekg(procStart, std::ios::beg);
125         file1.read(procBuffer1.data(), procSize);
126         file2.seekg(procStart, std::ios::beg);
127         file2.read(procBuffer2.data(), procSize);
128
129         MPI_Send(procBuffer1.data(), procSize, MPI_CHAR, i, 0, MPI_COMM_WORLD);
130         MPI_Send(procBuffer2.data(), procSize, MPI_CHAR, i, 0, MPI_COMM_WORLD);
131     }
132
133     // Корневой процесс тоже обрабатывает свой чанк
134     file1.seekg(0, std::ios::beg);
135     file1.read(buffer1.data(), chunkSize);
136     file2.seekg(0, std::ios::beg);
137     file2.read(buffer2.data(), chunkSize);
138 }
139 else
140 {
141     MPI_Recv(buffer1.data(), chunkSize, MPI_CHAR, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
142     MPI_Recv(buffer2.data(), chunkSize, MPI_CHAR, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
143 }
144
145 // Все машины производят операцию сравнения для своих чанков
146 bool isEqual = true;
147
148 #pragma omp parallel for reduction(&&:isEqual)
149 for (int i = 0; i < chunkSize; ++i) {
150     if (buffer1[i] != buffer2[i]) {
151         isEqual = false;
152     }
153 }
154
155 // Агрегируем результаты с помощью logical and
156 bool globalEqual;
157 MPI_Reduce(&isEqual, &globalEqual, 1, MPI_C_BOOL, MPI_LAND, 0,
MPI_COMM_WORLD);
158
159 if (rank == 0)
160 {
161     return globalEqual;
162 }
163
164 return true;
165 }
166
167 void calculateCompressionRatio(const string &originalFile, const string &compressedFile)
168 {

```

```

169     ifstream original(originalFile, ios::binary | ios::ate);
170     ifstream compressed(compressedFile, ios::binary | ios::ate);
171
172     if (!original.is_open() || !compressed.is_open())
173     {
174         cout << "Ошибка_при_открытии_файлов_для_вычисления_коэффициента_сжатия." <<
            endl;
175         return;
176     }
177
178     auto originalSize = original.tellg();
179     auto compressedSize = compressed.tellg();
180
181     if (originalSize == 0)
182     {
183         cout << "Ошибка:_размер_исходного_файла_равен_0." << endl;
184         return;
185     }
186
187     double compressionRatio = static_cast<double>(originalSize) / static_cast<double>(
        compressedSize);
188
189     cout << "Размер_исходного_файла:_ " << originalSize << "_байт" << endl;
190     cout << "Размер_сжатого_файла:_ " << compressedSize << "_байт" << endl;
191     cout << "Коэффициент_сжатия:_ " << compressionRatio << endl;
192 }
193

```

Приложение С1. Исходный код LZ77.h

```

1  #ifndef LZ77_H
2  #define LZ77_H
3
4  #include <string>
5  #include <vector>
6  #include <stdint>
7
8  class LZ77 {
9  public:
10     struct Token {
11         std::uint16_t offset;
12         std::uint16_t length;
13         char nextChar;
14
15         Token(std::uint16_t o = 0, std::uint16_t l = 0, char c = 0) : offset(o), length(l), nextChar(c) {}
16     };
17
18     LZ77();
19     void encodeFileMPI(const std::string& inputFileName, const std::string& outputFileName, int rank,
        int size);
20     void decodeFileMPI(const std::string& inputFileName, const std::string& outputFileName, int rank,
        int size);
21

```

```

22     std::vector<Token> encode(const std::string& input);
23     std::string decode(const std::vector<Token>& tokens);
24
25 private:
26     int WINDOW_SIZE;
27     int LOOKAHEAD_SIZE;
28 };
29
30 #endif // LZ77_H
31

```

Приложение С2. Исходный код LZ77.cpp

```

1  #include "LZ77.h"
2  #include <fstream>
3  #include <iostream>
4  #include <algorithm>
5  #include <cstring>
6  #include <mpi.h>
7
8  LZ77::LZ77() : WINDOW_SIZE(4096), LOOKAHEAD_SIZE(18) {}
9
10 std::vector<LZ77::Token> LZ77::encode(const std::string& input) {
11     std::vector<Token> tokens;
12     size_t pos = 0;
13
14     while (pos < input.size()) {
15         int bestLength = 0;
16         int bestOffset = 0;
17
18         int windowStart = std::max(0, static_cast<int>(pos) - WINDOW_SIZE);
19         int maxLength = std::min(LOOKAHEAD_SIZE, static_cast<int>(input.size() - pos));
20
21         for (int offset = 1; offset <= pos - windowStart; ++offset) {
22             int length = 0;
23             while (length < maxLength && input[pos - offset + length] == input[pos + length]) {
24                 ++length;
25             }
26
27             if (length > bestLength) {
28                 bestLength = length;
29                 bestOffset = offset;
30             }
31         }
32
33         char nextChar = (pos + bestLength < input.size()) ? input[pos + bestLength] : '\0';
34         tokens.emplace_back(static_cast<uint16_t>(bestOffset), static_cast<uint16_t>(bestLength),
35                             nextChar);
36         pos += bestLength + 1;
37     }
38     return tokens;
39 }

```

```

40
41 std::string LZ77::decode(const std::vector<Token>& tokens) {
42     std::string output;
43     for (const auto& token : tokens) {
44         int start = static_cast<int>(output.size()) - token.offset;
45         for (int i = 0; i < token.length; ++i)
46             output += output[start + i];
47         if (token.nextChar != '\0')
48             output += token.nextChar;
49     }
50     return output;
51 }
52
53 // ===== СЕРИАЛИЗАЦИЯ / ДЕСЕРИАЛИЗАЦИЯ =====
54
55 static std::vector<char> serializeTokens(const std::vector<LZ77::Token>& tokens) {
56     std::vector<char> buffer(tokens.size() * sizeof(LZ77::Token));
57     std::memcpy(buffer.data(), tokens.data(), buffer.size());
58     return buffer;
59 }
60
61 static std::vector<LZ77::Token> deserializeTokens(const std::vector<char>& buffer) {
62     size_t count = buffer.size() / sizeof(LZ77::Token);
63     std::vector<LZ77::Token> tokens(count);
64     std::memcpy(tokens.data(), buffer.data(), buffer.size());
65     return tokens;
66 }
67
68 // ===== ENCODE MPI =====
69
70 void LZ77::encodeFileMPI(const std::string& inputFileName, const std::string& outputFileName, int
    rank, int size) {
71     std::vector<char> inputBuffer;
72     int fullSize = 0;
73
74     if (rank == 0) {
75         std::ifstream in(inputFileName, std::ios::binary);
76         if (!in) {
77             std::cerr << "Failed_to_open_input_file\n";
78             MPI_Abort(MPI_COMM_WORLD, 1);
79         }
80         inputBuffer.assign(std::istreambuf_iterator<char>(in), {});
81         fullSize = inputBuffer.size();
82     }
83
84     MPI_Bcast(&fullSize, 1, MPI_INT, 0, MPI_COMM_WORLD);
85     inputBuffer.resize(fullSize);
86     MPI_Bcast(inputBuffer.data(), fullSize, MPI_CHAR, 0, MPI_COMM_WORLD);
87
88     // ==== Разделение данных по кускам ====
89     std::vector<int> counts(size), displs(size);
90     int base = fullSize / size, rem = fullSize % size;
91
92     for (int i = 0; i < size; ++i) {
93         counts[i] = base + (i < rem ? 1 : 0);
94         displs[i] = (i > 0 ? displs[i - 1] + counts[i - 1] : 0);

```

```

95     }
96
97     std::vector<char> localInput(counts[rank]);
98     MPI_Scatterv(inputBuffer.data(), counts.data(), displs.data(), MPI_CHAR,
99                 localInput.data(), counts[rank], MPI_CHAR, 0, MPI_COMM_WORLD);
100
101     // ===== Кодирование =====
102     std::string localStr(localInput.begin(), localInput.end());
103     std::vector<Token> localTokens = encode(localStr);
104     std::vector<char> serializedLocal = serializeTokens(localTokens);
105     int localSize = serializedLocal.size();
106
107     // ===== Сборка размеров =====
108     std::vector<int> recvSizes(size);
109     MPI_Gather(&localSize, 1, MPI_INT, recvSizes.data(), 1, MPI_INT, 0, MPI_COMM_WORLD);
110
111     std::vector<int> displsOut(size);
112     int totalOut = 0;
113     if (rank == 0) {
114         for (int i = 0; i < size; ++i) {
115             displsOut[i] = totalOut;
116             totalOut += recvSizes[i];
117         }
118     }
119     std::vector<char> gathered(totalOut);
120     MPI_Gatherv(serializedLocal.data(), localSize, MPI_CHAR,
121                 gathered.data(), recvSizes.data(), displsOut.data(), MPI_CHAR, 0,
122                 MPI_COMM_WORLD);
123
124     if (rank == 0) {
125         std::ofstream out(outputFileName, std::ios::binary);
126         out.write(gathered.data(), gathered.size());
127     }
128
129     // ===== DECODE MPI =====
130
131     void LZ77::decodeFileMPI(const std::string& inputFileName, const std::string& outputFileName, int
132                             rank, int size) {
133         std::vector<char> rawData;
134         int fullSize = 0;
135
136         if (rank == 0) {
137             std::ifstream in(inputFileName, std::ios::binary);
138             if (!in) {
139                 std::cerr << "Failed_to_open_compressed_file\n";
140                 MPI_Abort(MPI_COMM_WORLD, 1);
141             }
142             rawData.assign(std::istreambuf_iterator<char>(in), {});
143             fullSize = rawData.size();
144         }
145
146         MPI_Bcast(&fullSize, 1, MPI_INT, 0, MPI_COMM_WORLD);
147         rawData.resize(fullSize);
148         MPI_Bcast(rawData.data(), fullSize, MPI_CHAR, 0, MPI_COMM_WORLD);

```

```
149    // Только rank 0 декодирует чтобы избежать конфликтов)
150    if (rank == 0) {
151        auto tokens = deserializeTokens(rawData);
152        std::string result = decode(tokens);
153        std::ofstream out(outputFileName, std::ios::binary);
154        out.write(result.data(), result.size());
155    }
156
157    MPI_Barrier(MPI_COMM_WORLD);
158 }
159
```