



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования

«Дальневосточный федеральный университет»

(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент информационных и компьютерных систем

Никитин Даниил Олегович

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Выпускная квалификационная работа бакалавра

РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМОВ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ИЗОБРАЖЕНИЙ ДЛЯ СУПЕРКОМПЬЮТЕРА

по направлению подготовки 09.03.02 Информационные системы и технологии,
профиль «Информационные системы и технологии»

Владивосток
2022

В материалах данной выпускной
квалификационной работы не содержатся
сведения, составляющие государственную
тайну, и сведения, подлежащие экспертному
контролю

Уполномоченный по экспертному контролю

подпись

и. о. фамилия

« ____ » _____ 2022 г.


Защищена в ГЭК с оценкой

Секретарь ГЭК

подпись

И.О. Фамилия

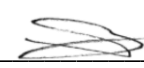
« ____ » _____ 2022 г.

Автор работы  _____ ,
подпись

студент группы Б9118-09.03.02 ист

« ____ » _____ 2022 г.

Руководитель ВКР Директор департамента,
д.ф.-м.н., профессор _____
должность, ученое звание

 _____ Е. В. Пустовалов
подпись и. о. фамилия

« ____ » _____ 2022 г.

«Допустить к защите»

Зав. кафедрой/ директор Департамента

д.ф.-м.н., профессор _____
ученая степень, ученое звание

 _____ Е. В. Пустовалов
подпись и. о. фамилия

« ____ » _____ 2022 г.

Оглавление

| | |
|---|----|
| Реферат | 4 |
| Введение | 5 |
| 1 Анализ предметной области..... | 7 |
| 1.1 Обоснование необходимости разработки программы для параллельной обработки изображений..... | 7 |
| 1.2 Технологии параллельного программирования | 7 |
| 1.3 Требования к разрабатываемой программе | 12 |
| 2 Анализ используемых технологий и их альтернатив | 14 |
| 2.1 Подходы в разработке | 14 |
| 2.2 Алгоритмы обработки данных | 15 |
| 2.3 Синтаксические анализаторы..... | 17 |
| 2.4 Обоснование выбора технологии параллельного программирования MPI..... | 21 |
| 3 Разработка программы и ее тестирование | 22 |
| 3.1 Разработка программы..... | 22 |
| 3.2 Преимущества разработанной программы | 43 |
| 3.3 Тестирование программы | 43 |
| Заключение..... | 46 |
| Список литературы | 48 |
| Приложение А Код программы | 51 |

Выпускная квалификационная работа «Разработка и реализация алгоритма параллельной обработки изображений для суперкомпьютера».

Автор работы – Никитин Даниил Олегович.

Научный руководитель ВКР – Пустовалов Евгений Владиславович.

Год защиты ВКР – 2022 г.

Реферат

Выпускная квалификационная работа – 71 с., 1 формула, 28 рис., 2 табл., 32 источника. Объектом исследования является параллельная обработка данных.

Цель работы – разработка алгоритма и программы для параллельной обработки изображений.

Предмет исследования – разработка алгоритма и программы для параллельной обработки графов.

В процессе выполнения работы была исследована область параллельной обработки данных, а конкретно, данных, полученных путем обработки изображений. По результатам анализа была выявлена необходимость создания алгоритма и программы для параллельной обработки изображений, умеющего параллельно считывать подготовленные данные из текстовых файлов, обрабатывать их, создавая из них 3 вида объектов, таких как графы, вершины и соседи вершин, и, затем, упорядоченно выводить полученные объекты в файлы с расширением .xml. Помимо внедренной в программу самой быстрой на сегодняшний день технологии распараллеливания и уникального алгоритма, разработанная программа имеет широкий набор функций, описывающих параметры каждого объекта.

В результате проведенной работы – разработана программа для параллельной обработки изображений, которая позволяет выполнять обработку данных с максимально возможной на сегодняшний день скоростью.

Разобранная по шагам работа реализованной программы дает понимание функциональности реализованной программы. Исходя из этого, главная цель выпускной квалификационной работы была выполнена.

Введение

Параллельная обработка оказывает огромное влияние на многие области компьютерных приложений. Растущее число приложений в науке, технике, бизнесе и медицине требует вычислительных скоростей, которые вряд ли могут быть достигнуты с помощью современного обычного компьютера. Эти приложения включают обработку огромного количества данных или выполнение большого количества итераций. Параллельная обработка — один из известных сегодня подходов, который поможет сделать эти вычисления выполнимыми. Вышеупомянутые приложения представляют множество ситуаций, в которых вероятность успешного выполнения вычислительной задачи увеличивается за счет использования компьютера, использующего параллельную обработку [1]. Разработанная программа настроена на работу с файлами, находящимися в одной хранилище.

Объектом исследования является разработка алгоритма для параллельной обработки изображений. Предметом исследования является разработка алгоритма для параллельной обработки графов.

Целью данной ВКР является разработка программы для параллельной обработки изображений. Для выполнения поставленной цели требуется решить ряд задач:

- реализовать класс «граф»;
- реализовать класс «вершина»;
- реализовать класс «сосед»;
- разработать алгоритм для обработки входных данных;
- реализовать вывод упорядоченных объектов, полученных из входных данных, в файлы с расширением .xml.
- реализовать параллельные чтение файлов, обработку данных и вывод в файлы.

Методами проектного исследования данной работы являются теоретический анализ, изучение соответствующей литературы, сравнение, проектирование.

В первой главе проведен анализ предметной области, включающий обзор технологий параллельных вычислений и требования к разрабатываемой программе.

Во второй главе представлены технологии, используемые для решения поставленных задач и рассматриваются их альтернативы.

В третьей главе описываются разработка программы, преимущества программы перед другими подобными приложениями и представлены результаты тестирования двух версий готовой программы, последовательной и параллельной.

1 Анализ предметной области

1.1 Обоснование необходимости разработки программы для параллельной обработки изображений

Программы, умеющие параллельно обрабатывать изображения, существуют, но каждая из них разрабатывалась для определенной области деятельности, все они работают с разными данными, по-разному их обрабатывают и, вообще, имеют отличный друг от друга функционал.

Департамент информационных и компьютерных систем ДВФУ заинтересован в разработке программы с конкретным набором функций и возможностей, которые отсутствуют у существующих приложений, поэтому была разработана данная программа.

Разработанная программа работает с номерами и координатами вершин (из которых состоят изображения), обрабатывает их и на их основе создает 3 вида объектов, которые затем упорядоченно выводит в xml файл. Одновременно описанные действия происходят с четырьмя файлами. В конечных файлах изначальные хаотичные данные представляются в виде упорядоченных списков графов, вершин и соседей вершин, а также у каждого объекта указаны свои параметры. Другими словами, данная программа преобразует данные, полученные от исходных изображений в списки графов.

1.2 Технологии параллельного программирования

1.2.1 POSIX Threads

Потоки POSIX или Pthreads — это переносимая библиотека потоков, которая обеспечивает согласованный программный интерфейс для нескольких операционных систем. Pthreads появился как стандартный интерфейс многопоточности для платформ Linux и Unix. Ряд поставщиков предоставляют API-интерфейсы потоков для конкретных поставщиков. Больше основных

функций Pthreads по созданию и уничтожению потоков, синхронизации и другим функциям потоков. В многопроцессорных архитектурах с общей памятью, таких как SMP, потоки могут использоваться для реализации параллелизма. Исторически сложилось так, что поставщики аппаратного обеспечения реализовывали свои собственные проприетарные версии потоков, что создавало проблему переносимости для разработчиков программного обеспечения.

Поток — это «облегченный процесс». Поток — это поток инструкций, который может быть запланирован как независимая единица. Поток существует внутри процесса и использует ресурсы процесса. Поскольку потоки очень малы по сравнению с процессами, создание потоков относительно дешево с точки зрения затрат на ЦП. Поскольку процессам требуется собственный пакет ресурсов, а потокам совместно используются ресурсы, потоки также экономят память. В процессе может быть несколько потоков. Многопоточные программы могут иметь несколько потоков, выполняющихся по разным путям кода одновременно [2].

1.2.2 OpenMP

OpenMP (Open Multi-Processing) — это интерфейс прикладного программирования (API), который поддерживает мультиплатформенное многопроцессорное программирование с общей памятью на C, C++ и Fortran на многих платформах, архитектурах с набором инструкций и операционных системах, включая Solaris, AIX, FreeBSD, HP-UX, Linux, macOS и Windows [3]. Он состоит из набора директив компилятора, подпрограмм библиотеки и переменных среды, влияющих на поведение во время выполнения [4].

OpenMP управляется некоммерческим технологическим консорциумом OpenMP Architecture Review Board (или OpenMP ARB), совместно определяемым широким кругом ведущих поставщиков компьютерного оборудования и программного обеспечения, включая Arm, AMD, IBM, Intel, Cray, HP, Fujitsu, Nvidia, NEC, Red Hat, Texas Instruments и Oracle Corporation [5].

OpenMP использует переносимую, масштабируемую модель, которая предоставляет программистам простой и гибкий интерфейс для разработки параллельных приложений для различных платформ, от стандартного настольного компьютера до суперкомпьютера [6].

1.2.3 PVM

Параллельная виртуальная машина (PVM) — это программный инструмент для параллельного объединения компьютеров в сеть. Он предназначен для использования сети разнородных машин Unix и/или Windows в качестве единого распределенного параллельного процессора. Таким образом, большие вычислительные задачи могут быть решены с меньшими затратами за счет использования совокупной мощности и памяти многих компьютеров. Программное обеспечение очень портативно; исходный код, доступный бесплатно через netlib, был скомпилирован для всего, от ноутбуков до суперкомпьютеров Crays.

PVM позволяет пользователям использовать имеющееся у них компьютерное оборудование для решения гораздо более крупных задач с меньшими дополнительными затратами. PVM использовался как образовательный инструмент для обучения параллельному программированию, но также использовался для решения важных практических задач. Он был разработан Университетом Теннесси, Окриджской национальной лабораторией и Университетом Эмори. Первая версия была написана в ORNL в 1989 году, а после перезаписи Университетом Теннесси в марте 1991 года была выпущена версия 2. Версия 3 была выпущена в марте 1993 года и поддерживала отказоустойчивость и лучшую переносимость [7].

1.2.4 UPC

Unified Parallel C (UPC) — расширение языка программирования C, предназначенное для высокопроизводительных вычислений на

крупномасштабных параллельных машинах, в том числе с общим глобальным адресным пространством (SMP и NUMA) и с распределенной памятью (например, для кластеров). Программисту предоставляется одно разделенное глобальное адресное пространство, где общие переменные могут напрямую считываться и записываться любым процессором, но каждая переменная физически связана с одним процессором. UPC использует модель вычислений с одной программой и несколькими данными (SPMD), в которой уровень параллелизма фиксируется во время запуска программы, обычно с одним потоком выполнения на процессор.

Чтобы выразить параллелизм, UPC расширяет язык программирования общего назначения ISO C 99 следующими конструкциями:

- явно параллельная модель выполнения;
- общее адресное пространство (shared квалификатор хранилища) с локальными частями потока (обычные переменные);
- примитивы синхронизации и модель согласованности памяти;
- явные коммуникационные примитивы;
- примитивы управления памятью [8].

1.2.5 Intel Threading Building Blocks

OneAPI Threading Building Blocks (oneTBB; ранее Threading Building Blocks или TBB) — это библиотека шаблонов C++, разработанная Intel для параллельного программирования на многоядерных процессорах. При использовании TBB вычисления разбиваются на задачи, которые могут выполняться параллельно. Библиотека управляет и планирует потоки для выполнения этих задач [9].

Программа oneTBB создает, синхронизирует и уничтожает графы зависимых задач в соответствии с алгоритмами, то есть парадигмами параллельного программирования высокого уровня (также известными как алгоритмические скелеты). Затем задачи выполняются с учетом зависимостей графа. Этот подход объединяет TBB в семейство методов параллельного

программирования, направленных на отделение программирования от особенностей базовой машины [10].

1.2.6 Boost

Boost — это набор библиотек для языка программирования C++, обеспечивающий поддержку таких задач и структур, как линейная алгебра, генерация псевдослучайных чисел, многопоточность, обработка изображений, регулярные выражения и модульное тестирование. Он содержит 164 отдельные библиотеки (начиная с версии 1.76) [11].

Все библиотеки Boost находятся под лицензией Boost Software License, разработанной для того, чтобы Boost можно было использовать как с бесплатными, так и с проприетарными программными проектами. Многие из основателей Boost входят в комитет по стандартам C++, и несколько библиотек Boost были приняты для включения в Технический отчет C++ 1, стандарт C++11 (например, интеллектуальные указатели, потоки, регулярные выражения, `random`, `ratio`, `tuple`) и стандарт C++17 (например, файловая система, любая, необязательно, вариант, `string_view`) [12].

1.2.7 MPI

MPI — это коммуникационный протокол для программирования параллельных компьютеров. Поддерживается как двухточечная, так и коллективная связь. MPI представляет собой интерфейс прикладного программирования с передачей сообщений вместе с протоколом и семантическими спецификациями, определяющими, как его функции должны вести себя в любой реализации. Целями MPI являются высокая производительность, масштабируемость и переносимость. MPI остается доминирующей моделью, используемой сегодня в высокопроизводительных вычислениях [13].

MPI не санкционирован ни одним крупным органом по стандартизации; тем не менее, он стал стандартом де-факто для связи между процессами, которые моделируют параллельную программу, работающую в системе с распределенной памятью. Настоящие суперкомпьютеры с распределенной памятью, такие как компьютерные кластеры, часто запускают такие программы [14].

1.3 Требования к разрабатываемой программе

Назначение разработки: параллельная обработка изображений.

Требования к программе:

- a. Программа по заданному пути к папке, должна считывать все имеющиеся в папке файлы и их обрабатывать.
- b. Алгоритм должен выделять группы точек по координатам XM, YM в отдельные графы.
- c. Условием выделения вершин в один граф является расстояние между вершинами, заданное в диапазоне значений от Rmin до Rmax.
- d. Графы должны быть оформлены в виде объектов соответствующего класса с указанным набором свойств и методов.
- e. По результатам обработки все графы, выделенные из одного файла, записываются в формате XML в один файл с исходным именем с добавлением расширения xml.
- f. Алгоритм должен выполнять параллельную или высокопроизводительную обработку данных с использованием потоков/ядер ЦПУ или ГПУ.

Требования к классу «граф/graph»:

- a. Класс должен иметь имя – порядковый номер, уникальный в пределах одного обрабатываемого файла данных – тип строка.
- b. Свойство энергия графа/Energy – тип реальное число двойной точности.
- c. Свойство список вершин – объектов класса Вершина/Vertex с соответствующими свойствами – тип список вершин.
- d. Методы:

- 1) получить число вершин в графе;
- 2) получить объект вершина по номеру вершины в графе;
- 3) получить/вычислить радиус графа;
- 4) получить/вычислить диаметр графа;
- 5) получить/вычислить длину графа;
- 6) получить список вершин в текстовом формате;
- 7) получить/вычислить энергию графа для заданного R0;
- 8) добавить объект вершина в список вершин графа;
- 9) стандартный конструктор и деструктор.

Требования к классу вершина/vertex:

а. Свойства:

- 1) имя вершины – номер вершины из исходного файла – тип текст/строка;
- 2) номер в графе – порядковый номер вершины в графе – тип целое;
- 3) координата X – координата вершины из исходного файла – реальное двойной точности;
- 4) координата Y – координата вершины из исходного файла – реальное двойной точности;
- 5) список соседей вершины – список объектов класса сосед/Neighbor.

б. Методы:

- 1) получить число соседей;
- 2) добавить соседа;
- 3) получить объект сосед по номеру в списке;
- 4) стандартный конструктор и деструктор.

Требования к классу сосед/Neighbor:

а. Свойства:

- 1) имя – номер вершины из исходного файла – тип текст/строка;
- 2) номер в графе - порядковый номер вершины в графе – тип целое.

б. Стандартный конструктор и деструктор.

2 Анализ используемых технологий и их альтернатив

2.1 Подходы в разработке

2.1.1 Процедурный подход

Этот подход также известен как подход «сверху вниз». При таком подходе программа делится на функции, выполняющие определенные задачи. Этот подход в основном используется для приложений среднего размера. Данные являются глобальными, и все функции могут обращаться к глобальным данным. Основным недостатком процедурного подхода к программированию является то, что данные не защищены, поскольку они являются глобальными и доступны для любой функции. Поток управления программой достигается посредством вызовов функций и операторов goto. Данный подход используется при работе с языками программирования C, FORTRAN и COBOL [15].

2.1.2 Объектно-ориентированный подход

Основным мотивом изобретения объектно-ориентированного подхода (ООП) является устранение некоторых недостатков процедурного подхода. ООП рассматривает данные как важнейший элемент разработки программы и не позволяет им свободно перемещаться по системе. Он более тесно связывает данные с функциями, которые с ними работают, и защищает их от случайного изменения внешними функциями.

ООП позволяет разложить проблему на ряд сущностей, называемых объектами, а затем строить данные и функции вокруг этих объектов. Доступ к данным объекта возможен только с помощью функций, связанных с объектом. Однако функции одного объекта могут обращаться к функциям других объектов [16].

2.1.3 Преимущества ООП

Так как одним из основных требований к программе является наличие в ней трех реализованных классов и работа с объектами, а также, то, что код, в целом, становится значительно понятнее и проще при использовании ООП, то было решено использовать объектно-ориентированный подход. Помимо перечисленного, данный подход имеет следующие преимущества:

1. Добавлять новые данные и функции легко [17].
2. Уменьшенная сложность задачи. Данную задачу можно рассматривать как совокупность различных объектов. Каждый объект отвечает за определенную задачу.
3. Простота обслуживания и обновления: ООП упрощает обслуживание и изменение существующего кода, поскольку можно создавать новые объекты с небольшими отличиями от существующих.
4. Передача сообщений. Техника обмена сообщениями между объектами упрощает взаимодействие с внешними системами.
5. Модифицируемость: легко внести незначительные изменения в представление данных или процедуры в объектно-ориентированной программе. Изменения внутри класса не влияют ни на какую другую часть программы, поскольку единственный общедоступный интерфейс, который внешний мир имеет для класса, — это использование методов [16].

2.2 Алгоритмы обработки данных

2.2.1 Поиск в глубину

1. Поиск в глубину — это алгоритм обхода или поиска структур данных в виде дерева или графа [18]. Алгоритм начинается с корневого узла (выбирая какой-либо произвольный узел в качестве корневого узла в случае графа) и исследует как можно дальше каждую ветвь перед возвратом. Таким образом, основная идея состоит в том, чтобы начать с корня или любого произвольного

узла, пометить узел и перейти к соседнему непомеченному узлу и продолжать этот цикл до тех пор, пока не останется непомеченного соседнего узла. Затем алгоритм возвращается и проверяет другие непомеченные узлы и проходит их. Наконец, печатаются узлы пути [19].

2. Стандартная реализация данного алгоритма помещает каждую вершину графа в одну из двух категорий:

3. Посещена.

4. Не посещена.

5. Цель алгоритма — пометить каждую вершину как посещенную, избегая при этом циклов [20].

6. «Поиск в глубину» работает следующим образом:

1. Любая из вершин графа помещается на вершину стека.

2. Берется верхний элемент стека и добавляется в список посещенных.

3. Создается список смежных узлов этой вершины. Те, которых нет в списке посещенных, добавляются в верхнюю часть стека.

4. Шаги 2 и 3 повторяются, пока стек не станет пустым [21].

2.2.2 Поиск в ширину

Поиск в ширину — это алгоритм обхода, в котором обход начинается с выбранного узла (исходного или начального узла) и проходит по графу послойно, таким образом исследуя соседние узлы (узлы, которые напрямую связаны с исходным узлом). Затем происходит переход к соседним узлам следующего уровня [22].

Как следует из названия данного алгоритма, он проходит по графу в ширину следующим образом:

1. Сначала алгоритм перемещается по горизонтали и посещает все узлы текущего слоя.

2. Далее алгоритм переходит к следующему слою [23].

Стандартная реализация поиска в ширину помещает каждую вершину графа в одну из двух категорий:

1. Посещена.
2. Не посещена.

Цель алгоритма — пометить каждую вершину как посещенную, избегая при этом циклов.

Алгоритм работы поиска в ширину следующий:

1. Любая вершина графа помещается в конец очереди.
2. Берется первый элемент очереди и добавляется в список посещенных.
3. Создается список смежных узлов этой вершины. Те, которых нет в списке посещенных, добавляются в конец очереди.
4. Шаги 2 и 3 повторяются, пока очередь не станет пустой [24].

Граф может иметь две разные несвязанные части, поэтому, чтобы убедиться, что покрывается каждая вершина, можно запустить алгоритм поиска в ширину на каждом узле [25].

2.2.3 Преимущества поиска в ширину

Для разработки алгоритма обработки данных было решено выбрать поиск в ширину, так как он обладает следующими преимуществами перед поиском в глубину:

1. Поиск в ширину обязательно найдет решение, если оно есть.
2. Поиск в ширину никогда не застрянет в тупике, то есть в нежелательных узлах.
3. Если существует более одного решения, он найдет решение с минимальными шагами [26].

2.3 Синтаксические анализаторы

2.3.1 TinyXML

TinyXML — это простой, небольшой XML-парсер C++, который можно легко интегрировать в другие программы.

Вкратце, TinyXML анализирует XML-документ и строит на его основе объектную модель документа (DOM), которую можно читать, изменять и сохранять.

Существуют различные способы доступа и взаимодействия с XML-данными. TinyXML использует объектную модель документа (DOM), что означает, что данные XML анализируются в объекты C++, которые можно просматривать и манипулировать, а затем записывать на диск или в другой выходной поток. Вы также можете создать XML-документ с нуля с помощью объектов C++ и записать его на диск или в другой выходной поток.

TinyXML разработан таким образом, чтобы его было легко и быстро освоить. Это два заголовка и четыре файла `cpp`.

TinyXML пытается быть гибким синтаксическим анализатором, но с действительно правильным и совместимым выводом XML. TinyXML должен компилироваться в любой достаточно совместимой с C++ системе. Он не полагается на исключения или RTTI. Он может быть скомпилирован с поддержкой STL или без нее. TinyXML полностью поддерживает кодировку UTF-8 и первые 64 000 символов.

TinyXML поддерживает кодировку UTF-8, что позволяет работать с файлами XML на любом языке. TinyXML также поддерживает «устаревший режим» — кодировку, использовавшуюся до поддержки UTF-8 и, вероятно, лучше всего описываемую как «расширенный ascii» [27].

2.3.2 LibXML2

Написанная на языке программирования C, libxml2 обеспечивает привязку к C++, Ch, XSH, C#, Python, Kylix/Delphi и другим языкам Pascal, Ruby, Perl, Common Lisp, и PHP. Первоначально он был разработан для проекта GNOME, но может использоваться вне его. Код libxml2 очень переносим, так как он зависит только от стандартных ANSI C библиотек, и он выпущен под лицензией MIT. Libxml2 очень переносим, библиотека должна собираться и работать без

серьезных проблем на различных системах (Linux, Unix, Windows, CygWin, MacOS, RISC Os, OS/2, VMS, QNX, MVS, ...) [28].

Несколько ключевых моментов о libxml:

1. Libxml2 экспортирует интерфейсы синтаксического анализатора типа Push (прогрессивный) и Pull (блокирующий) как для XML, так и для HTML.
2. Libxml2 может выполнять проверку DTD (Document Type Definition – определение типа документа) во время синтаксического анализа, используя проанализированный экземпляр документа или с произвольным DTD.
3. Libxml2 включает полные реализации XPath 1.0, XPointer и XInclude 1.0.
4. Он написан на простом языке C, с минимальным количеством.
5. Строго придерживается ANSI C/POSIX для простоты встраивания. Работает на Linux/Unix/Windows, портирован на ряд других платформ.
6. Базовая поддержка клиентов HTTP и FTP, позволяющая приложениям получать удаленные ресурсы.
7. Конструкция модульная, большинство расширений можно компилировать.
8. Внутреннее представление документа максимально приближено к DOM - интерфейсам.
9. Libxml2 также имеет SAX-подобный интерфейс; интерфейс предназначен для совместимости с Expat.
10. Эта библиотека выпущена под лицензией MIT. Точную формулировку смотрите в файле Copyright в дистрибутиве [29].

2.3.3 PugiXML

Pugixml — это облегченная библиотека обработки XML на C++.

Особенности:

- DOM-подобный интерфейс с широкими возможностями обхода/модификации.

- Чрезвычайно быстрый синтаксический анализатор XML без проверки, который строит дерево DOM из файла/буфера XML.
- Реализация XPath 1.0 для сложных запросов дерева, управляемых данными.
- Полная поддержка Unicode с вариантами интерфейса Unicode и автоматическим преобразованием кодировки.

Библиотека чрезвычайно портативна и проста в интеграции и использовании [30].

2.3.4 RapidXML

RapidXml — это парсер in-situ, который позволяет достичь очень высокой скорости парсинга. In-situ означает, что синтаксический анализатор не делает копии строк. Вместо этого он помещает указатели на исходный текст в иерархию DOM [31].

2.3.5 Преимущества RapidXML

Для реализации вывода обработанных данных в XML файлы было решено выбрать RapidXML:

1. Весь синтаксический анализатор содержится в одном заголовочном файле, поэтому нет необходимости в сборке или компоновке.
2. Очень высокая скорость разбора. Как правило, скорость синтаксического анализа примерно в 50–100 раз выше, чем у Xerces DOM, в 30–60 раз быстрее, чем у TinyXml, в 3–12 раз быстрее, чем у pugxml, и примерно на 5–30% быстрее, чем у pugixml, самого быстрого из известных парсеров XML.
3. Позволяет изменять дерево DOM. Дерево DOM, созданное синтаксическим анализатором, полностью модифицируемо. Можно добавлять/удалять узлы и атрибуты, а также менять их содержимое[31].

2.4 Обоснование выбора технологии параллельного программирования MPI

Для реализации распараллеливания было решено выбрать технологию параллельных вычислений MPI, так как она имеет следующие преимущества перед остальными подобными технологиями:

- данная технология имеет самую высокую скорость;
- легко подключается к проекту;
- работает на архитектурах с общей или распределенной памятью;
- может использоваться для решения более широкого круга задач, чем OpenMP (главный конкурент по скорости);
- каждый процесс имеет свои локальные переменные;
- компьютеры с распределенной памятью дешевле, чем большие компьютеры с общей памятью;
- память масштабируется с количеством процессоров;
- каждый процессор может быстро обращаться к своей памяти без помех [32].

В одном месте кода была также использована технология OpenMP, но ее роль в данной программе слишком мала, чтобы много об этом писать. Она используется только для ускорения получения времени окончания каждого процесса, не считая нулевого.

3 Разработка программы и ее тестирование

3.1 Разработка программы

3.1.1 Проектирование программы

3.1.1.1 Блок-схема программы

Общая блок-схема работы всей программы без детализации представлена на рисунке 3.

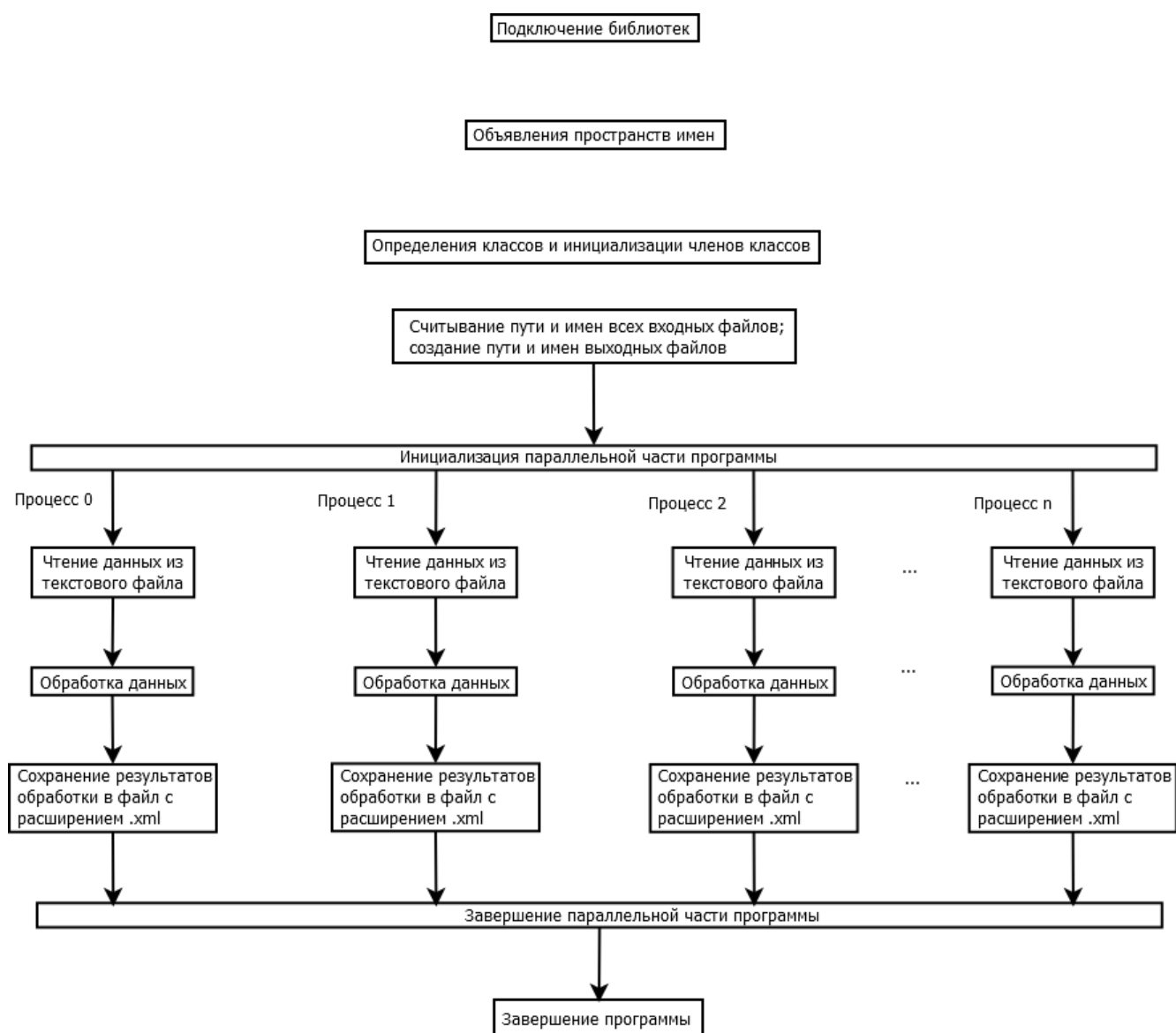


Рисунок 1 – Блок-схема программы

3.1.1.2 Диаграмма классов

Диаграмма разработанных классов содержит сами классы, их связи друг с другом и мощности данных связей, и представлена на рисунке 2.

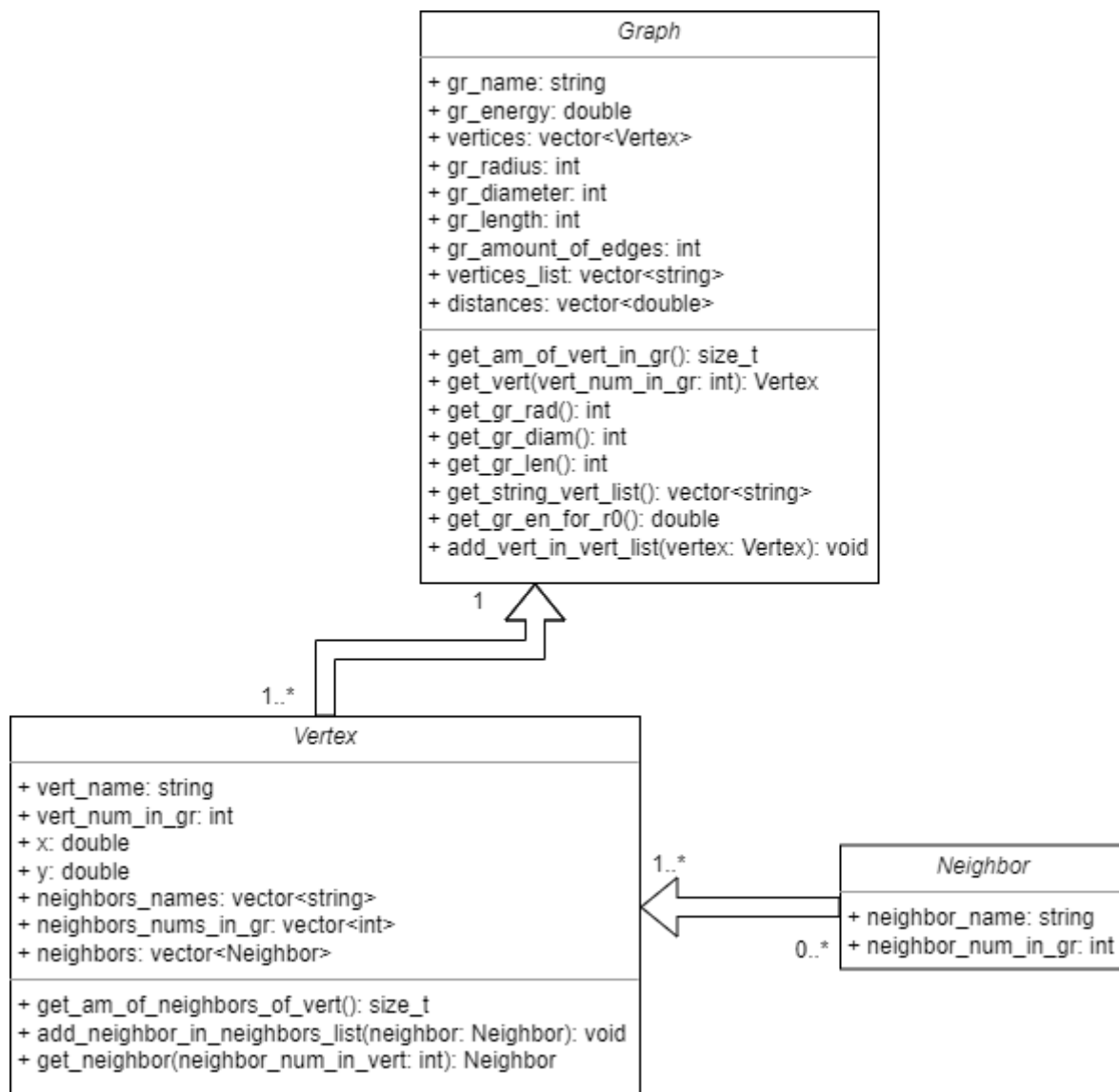


Рисунок 2 - Диаграмма классов

3.1.1.3 Блок-схема работы параллельных процессов

Блок-схема работы параллельных процессов включает только код, расположенный в параллельной части программы и показывает все, что происходит внутри каждого из порожденных процессов на примере одного из них. В данной блок-схеме акцент сделан на порождении параллельных процессов и алгоритме обработке изображений, но части со считыванием данных из файлов

и выводом в xml файлы показаны без детализации, каждая – в виде одного блока с небольшим описанием. Блок-схема по частям представлена на рисунках с 3 по 10.

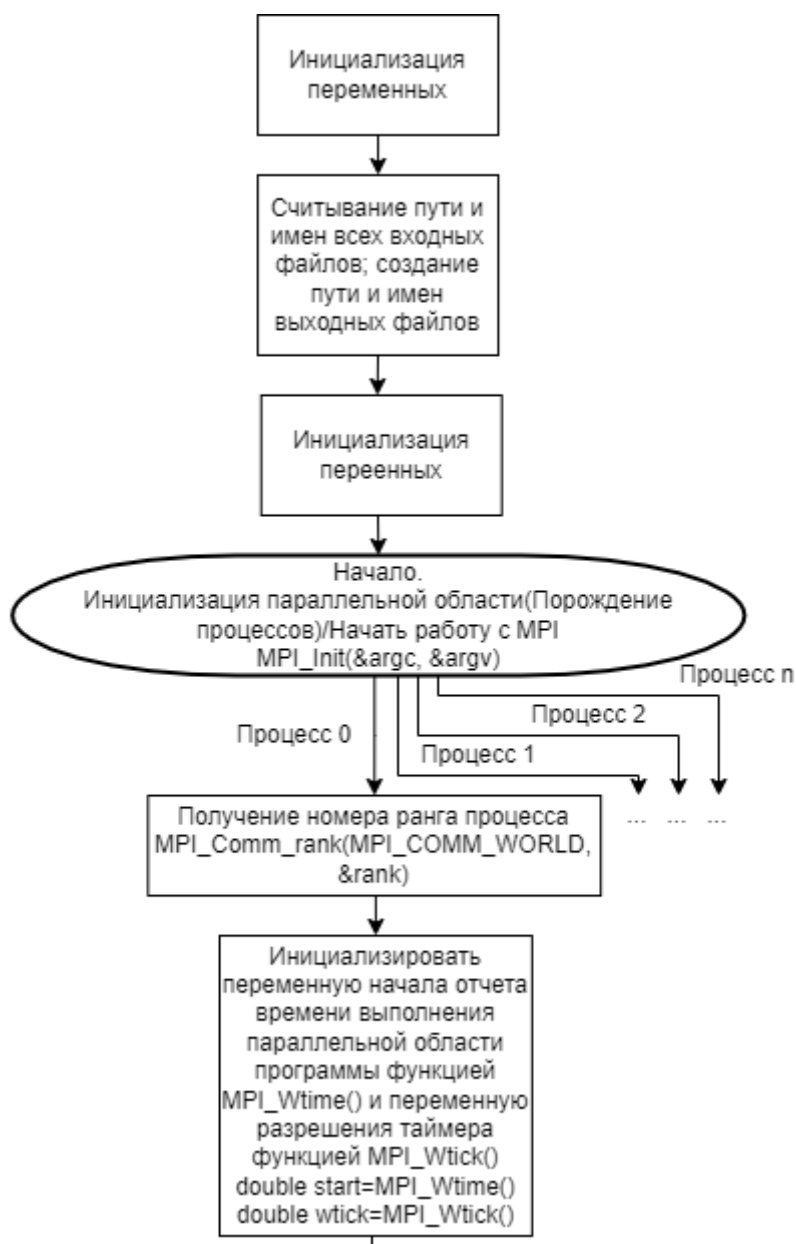


Рисунок 3 – Часть 1 блок-схемы работы одного из параллельных процессов



Рисунок 4 – Часть 2 блок-схемы работы одного из параллельных процессов

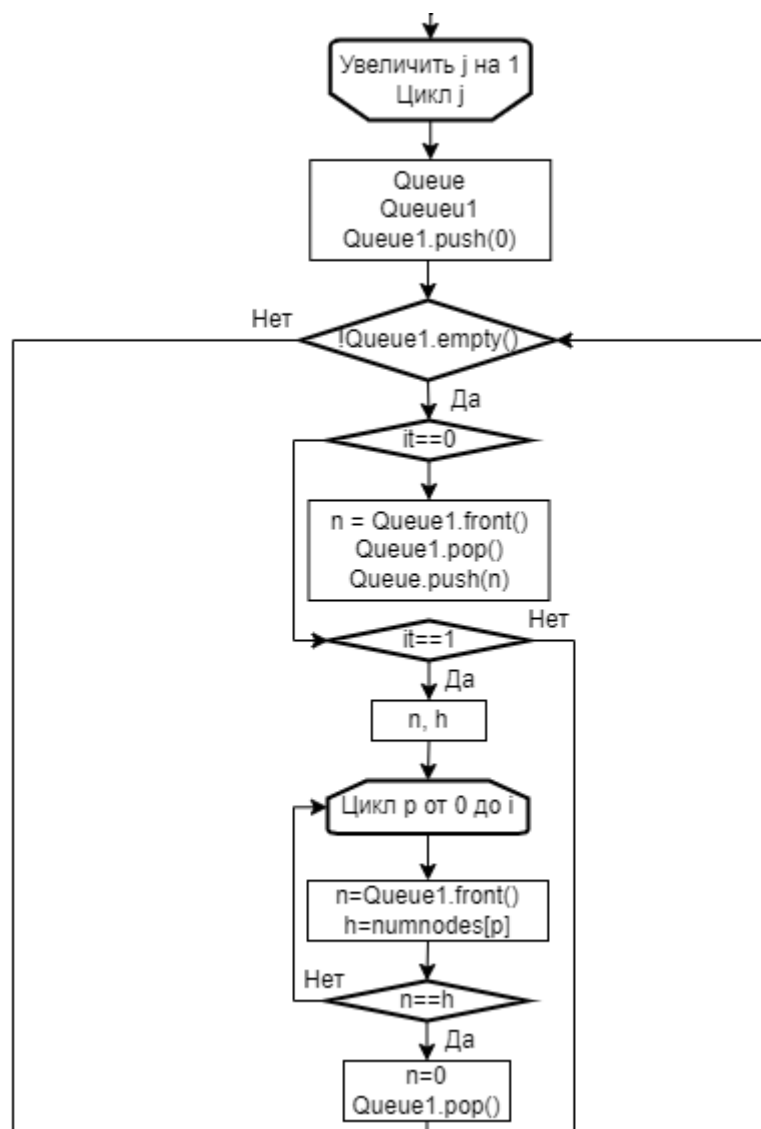


Рисунок 5 – Часть 3 блок-схемы работы одного из параллельных процессов

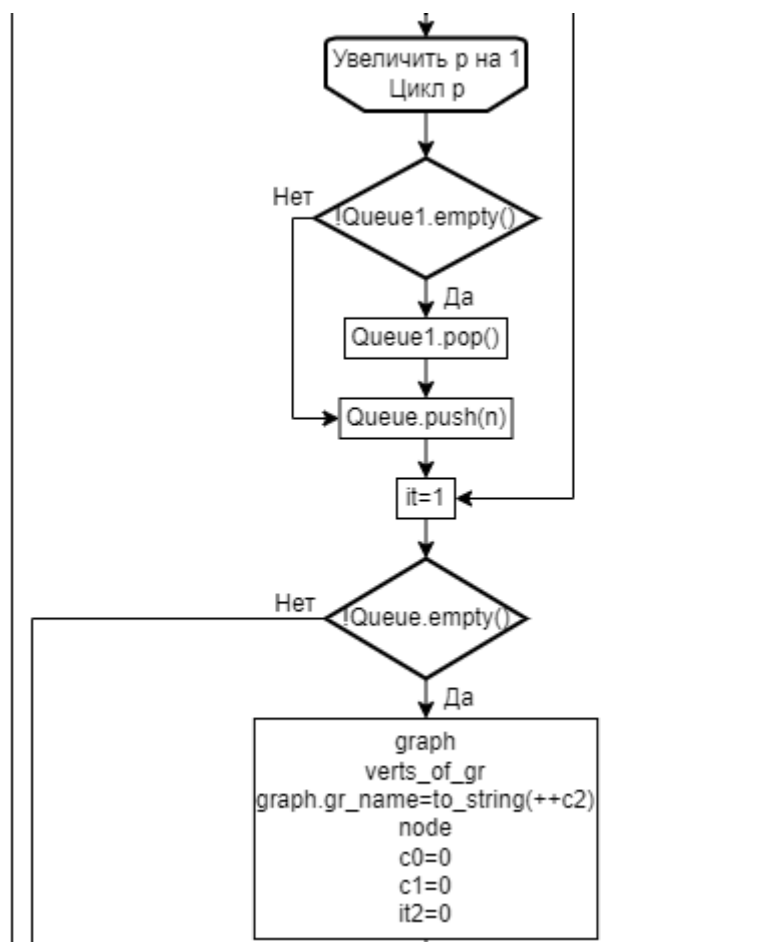


Рисунок 6 – Часть 4 блок-схемы работы одного из параллельных процессов

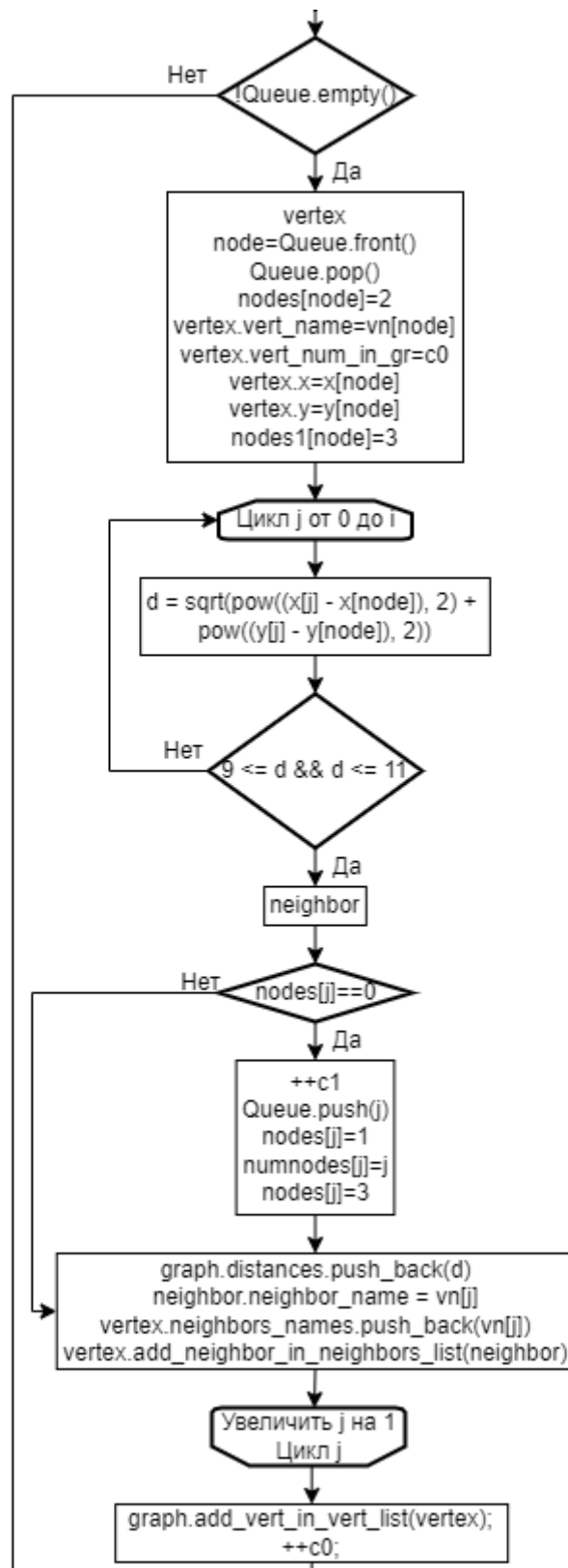


Рисунок 7 – Часть 5 блок-схемы работы одного из параллельных процессов

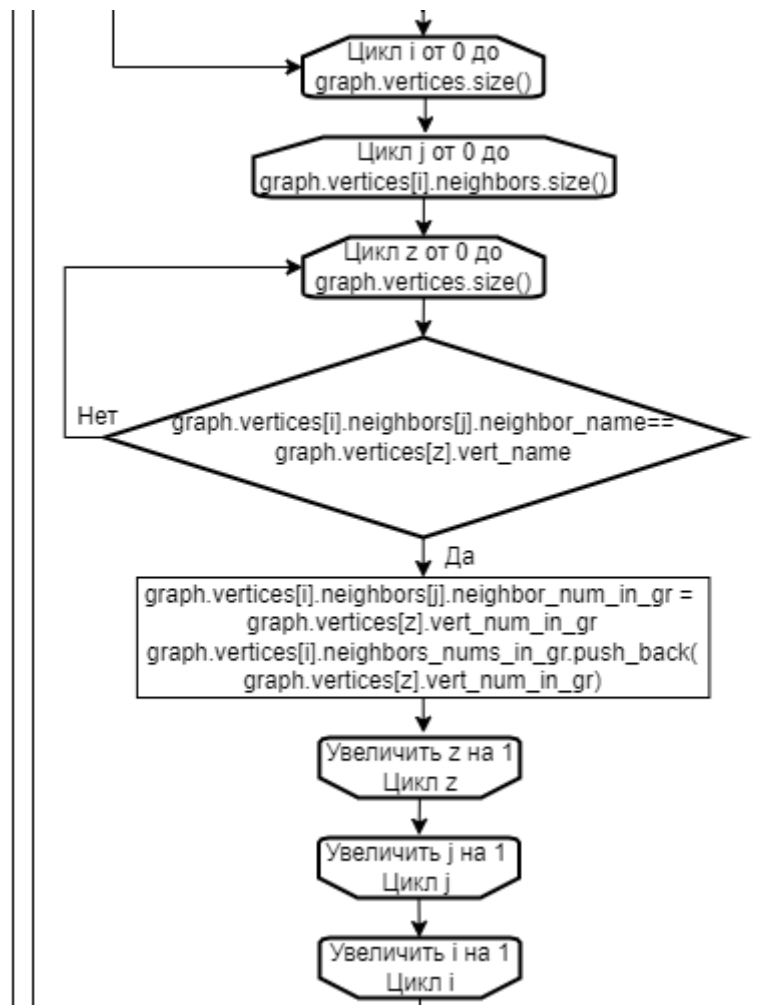


Рисунок 8 – Часть 6 блок-схемы работы одного из параллельных процессов

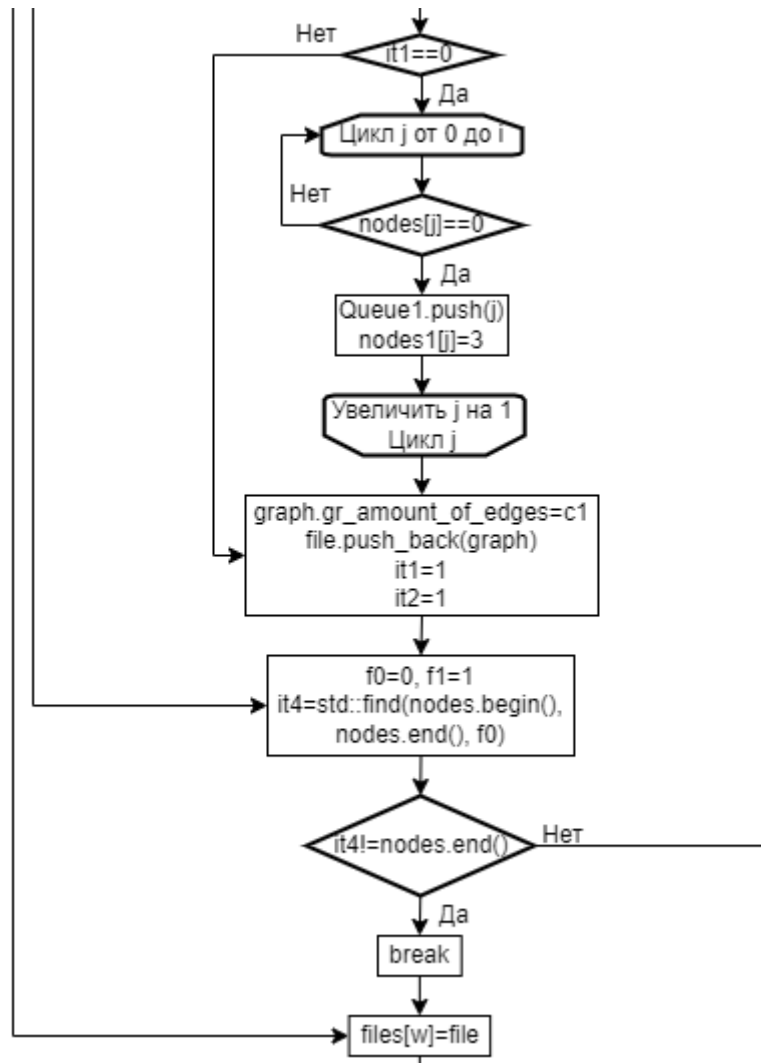


Рисунок 9 – Часть 7 блок-схемы работы одного из параллельных процессов

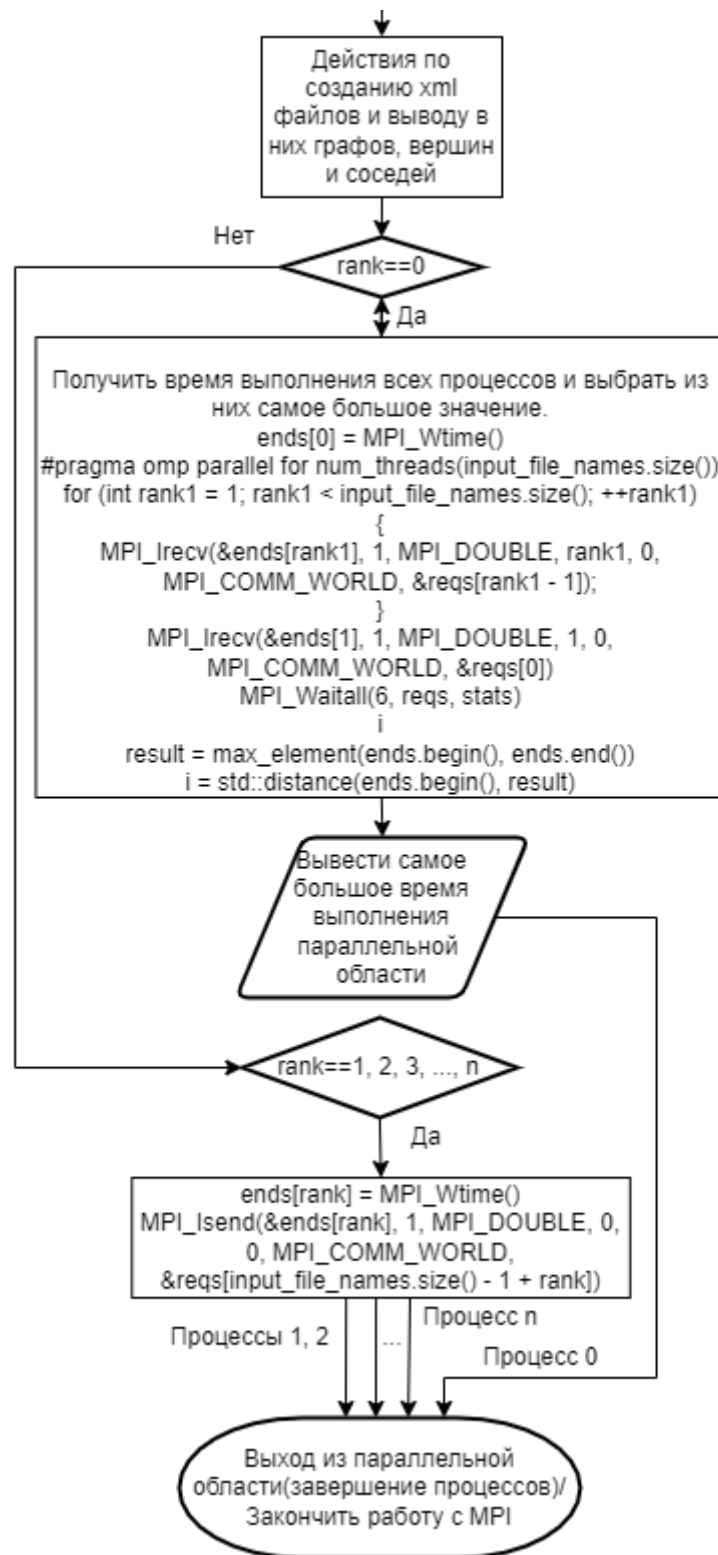


Рисунок 10 – Часть 8 блок-схемы работы одного из параллельных процессов

3.1.2 Реализация классов

Классы были реализованы с помощью механизма вложенности, так как это логически правильно и дает возможность создать между объектами классов

прямые связи. Так как каждый граф состоит из хотя бы одной вершины, то класс «граф» содержит класс «вершина». У каждой вершины может быть 0 или более соседей, следовательно, класс «вершина» содержит класс «сосед». Итого, класс «граф» содержит классы «вершина» и «сосед» и является для них объемлющим, а они для него – вложенными.

В каждом классе прописаны свои свойства и методы. Часть методов создана для вложения экземпляров одного класса в вектора других экземпляров. Другие методы предназначены для вычисления параметров объектов того класса, для которого эти методы написаны. Свойства делятся на те, которые хранят объекты других классов и на те, которые хранят свойства объектов своего класса.

Реализованные классы представлены на рисунках с 11 по 13.

```

25 class Graph
26 {
27 public:
28     Graph()
29     {
30     }
31 }
32 class Vertex
33 {
34 public:
35     Vertex()
36     {
37     }
38 }
39 class Neighbor
40 {
41 public:
42     Neighbor()
43     {
44     }
45 }
46     string vert_name;
47     int vert_num_in_gr;
48     double x, y;
49     vector<string> neighbors_names;
50     vector<int> neighbors_nums_in_gr;
51     vector<Neighbor> neighbors;
52     size_t get_am_of_neighbors_of_vert()//Vertex метод: Пор
53     {
54         return neighbors.size();
55     }
56     void add_neighbor_in_neighbors_list(Neighbor neighbor)/
57     {
58         neighbors.push_back(neighbor);
59     }
60     Neighbor get_neighbor(int neighbor_num_in_vert)//Vertex
61     {
62         return neighbors[neighbor_num_in_vert];
63     }
64     ~Vertex()
65     {
66     }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }

```

Рисунок 11 – Классы


```

70     }
71     ~Vertex()
72     {
73     }
74 }
75 };
76 string gr_name;
77 double gr_energy;
78 vector<Vertex> vertices;
79 int gr_radius, gr_diameter, gr_length, gr_amount_of_edges;
80 vector<string> vertices_list;
81 vector<double> distances;
82 size_t get_am_of_vert_in_gr()//Graph метод: Получить число вершин в граф
83 {
84     return vertices.size();
85 }
86 Vertex get_vert(int vert_num_in_gr)//Graph метод: Получить объект вершин
87 {
88     return vertices[vert_num_in_gr];
89 }
90 int get_gr_rad()//Graph метод: Получить/вычислить радиус графа
91 {
92     if (distances.size() > 0)
93     {
94         int i;
95         auto result1 = min_element(distances.begin(), distances.end());
96         i = std::distance(distances.begin(), result1);
97         gr_radius = distances[i];
98     }
99     else
100     {
101         gr_radius = 0;
102     }
103     return gr_radius;
104 }
105 int get_gr_diam()//Graph метод: Получить/вычислить диаметр графа
106 {
107     if (distances.size() > 0)
108     {
109         int i;
110         auto result2 = max_element(distances.begin(), distances.end());
111         i = std::distance(distances.begin(), result2);
112         gr_diameter = distances[i];
113     }
114     else
115     {
116         gr_diameter = 0;
117     }
118     return gr_diameter;
119 }

```

Рисунок 12 – Классы

```

118         return gr_diameter;
119     }
120     int get_gr_len()//Graph метод: Получить/вычислить длину графа
121     {
122         gr_length = gr_amount_of_edges;
123         return gr_length;
124     }
125     vector<string> get_string_vert_list()//Graph метод: Получить список вершин в текстовом
126     {
127         for (int i = 0; i < vertices.size(); ++i)
128         {
129             vertices_list.push_back(vertices[i].vert_name);
130         }
131         return vertices_list;
132     }
133     double get_gr_en_for_r0()//Graph метод: Получить/вычислить энергию графа для заданн
134     {
135         if (distances.size() > 0)
136         {
137             gr_energy = 0;
138             double r0 = 10.0;
139             for (int i = 0; i < distances.size(); ++i)
140             {
141                 gr_energy += pow((r0/distances[i]), 12) - pow((r0 / distances[i]), 6);
142             }
143             return gr_energy;
144         }
145         else
146         {
147             gr_energy = 0;
148         }
149         return gr_energy;
150     }
151     void add_vert_in_vert_list(Vertex vertex)//Graph метод: Добавить объект вершина в с
152     {
153         vertices.push_back(vertex);
154     }
155     ~Graph()
156     {
157     }
158 };
159
160
161 int main(int argc, char *argv[])
162 {
163     int rank, w, i = 0, it = 0, it1 = 0, c0, c1, c2 = 0, it2;
164     vector<double> ends(4), x, y;
165     vector<vector<Graph>> files(4);
166     vector<string> paths = { "d_10-Gauss filtered g=-16 series4_00001.mrc_sl_0_d_10_n_7:
167     vector<Graph> file:

```

Рисунок 13 - Классы

3.1.3 Разработка алгоритма обработки данных

Разработанный алгоритм поиска в ширину работает с конкретными данными, полученными из текстового файла и добавленными в соответствующие вектора, и в ходе обработки создает список графов, каждый из которых имеет свой список вершин, входящих в конкретный граф. Каждая вершина, в свою очередь, имеет список своих соседей. Ограничением, с помощью которого получают отдельные графы и находятся соседи вершин, является определенное расстояние от начальной вершины до ее возможных соседей, которое вычисляется по формуле (1)

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \quad (1)$$

где d – расстояние между двумя вершинами;

x_2, x_1, y_2, y_1 – координаты вершин, между которыми определяется расстояние.

Все виды объектов создаются каждый в своем циклы. После того, как найдутся все соседи последней вершины графа, он добавляется в вектор `file` и начинается следующая итерация цикла графов, в которой создается экземпляр класса «граф» и начинается поиск вершин и их соседей из оставшихся после первой итерации вершин. Цикл графов повторяется до тех пор, пока в векторах не останется данных для обработки. Далее начинается следующий этап работы программы, а именно создание `xml` файла и его заполнение упорядоченным списком полученных графов, вершин и соседей.

Код разработанного алгоритма представлен на рисунке А.1.

3.1.4 Реализация вывода обработанных данных в XML файлы

Подключение RapidXML показано на рисунках с 14 по 17.

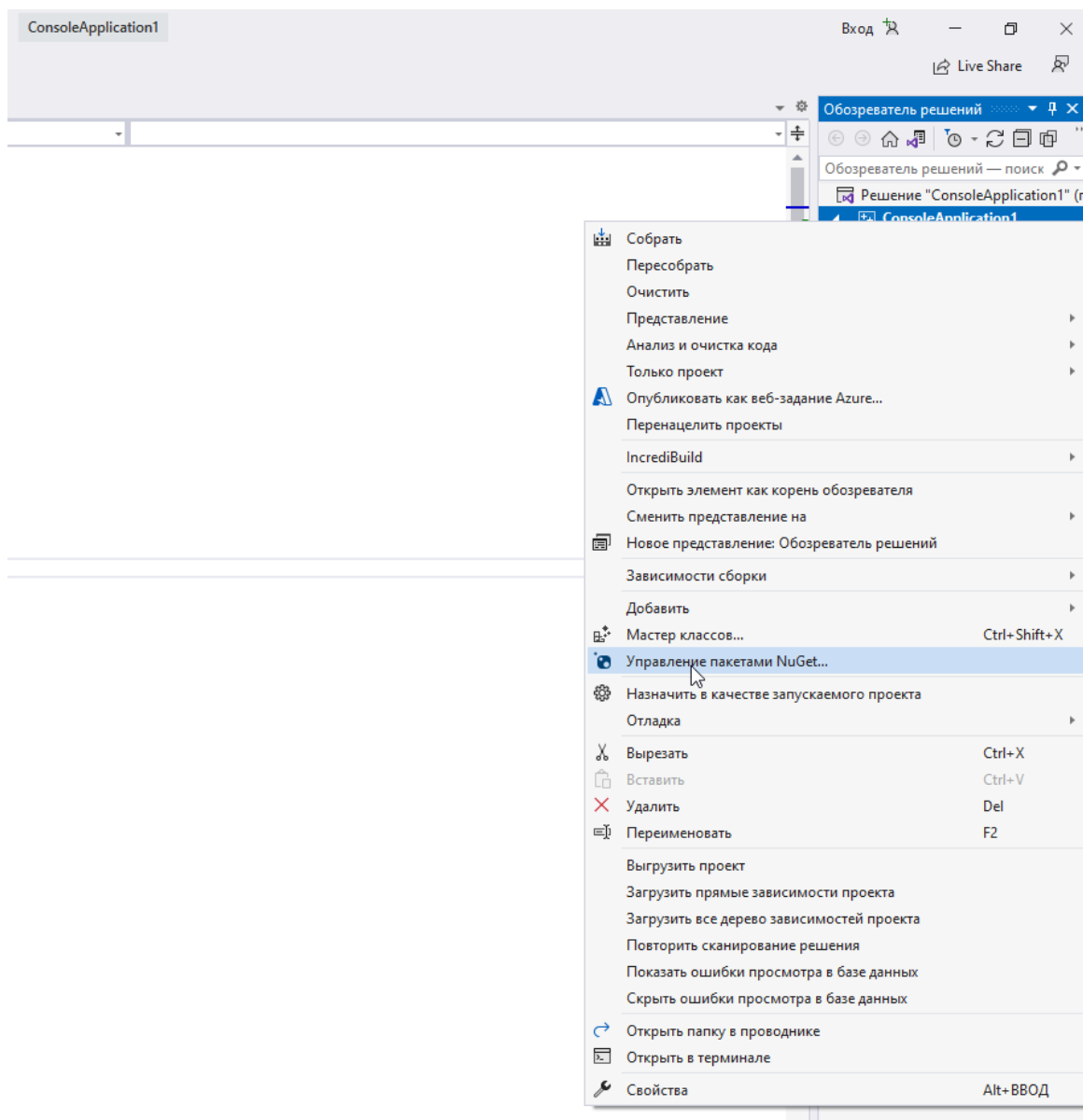


Рисунок 14 – Подключение RapidXML к проекту

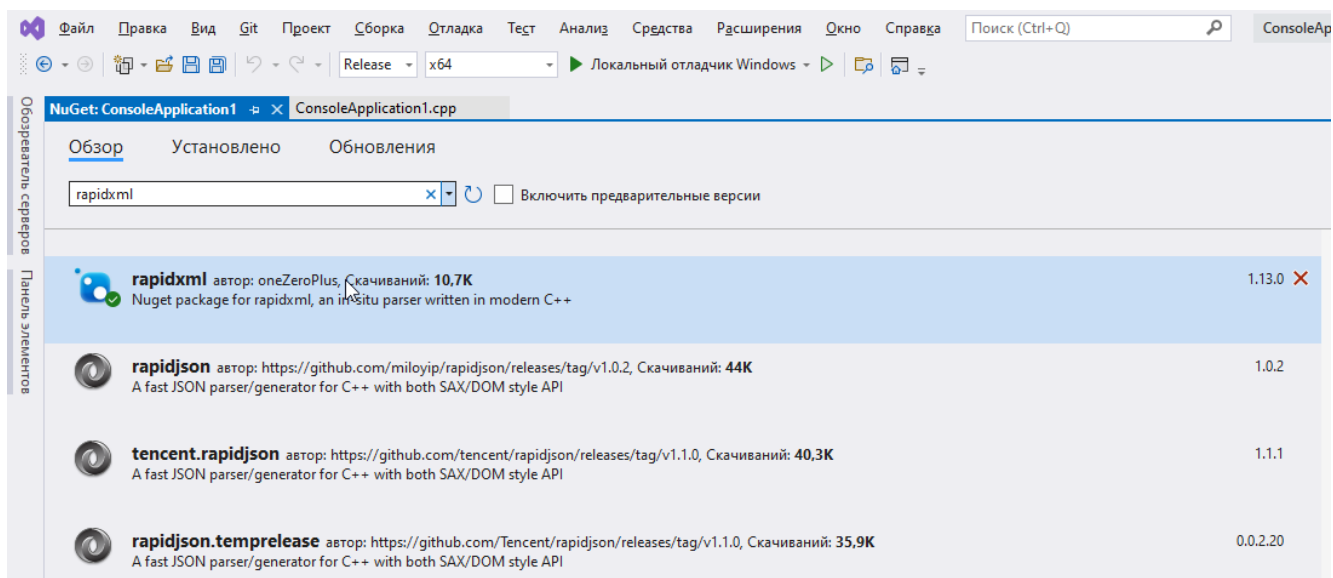


Рисунок 15 – Скачивание и установка RapidXML

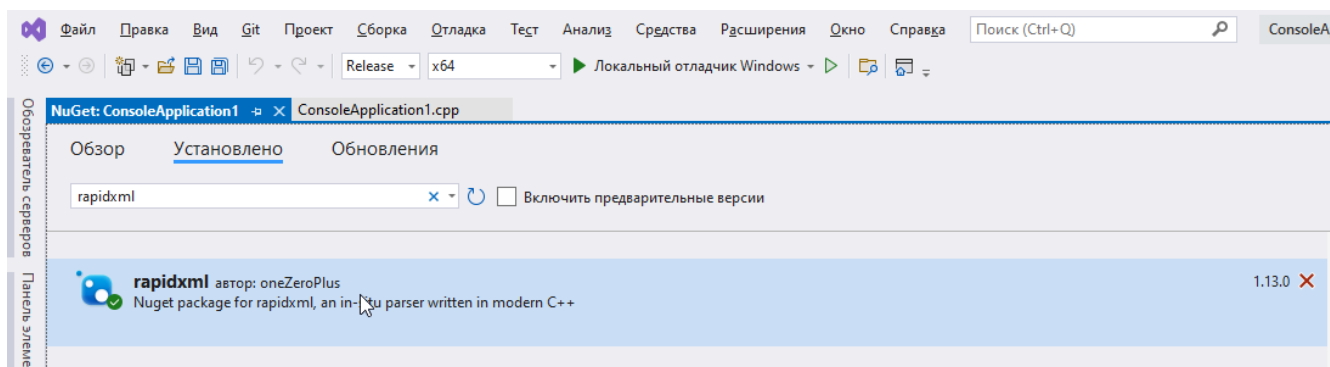


Рисунок 16 –RapidXML установлена

```

13  #include <rapidxml/rapidxml.hpp>
14  #include <rapidxml/rapidxml print.hpp>
15  #include <rapidxml/rapidxml iterators.hpp>
16  #include <rapidxml/rapidxml utils.hpp>
17  #include <iomanip>
18  #include <windows.h>
19  #include "mpi.h"
20
21  using std::string;
22  using namespace std;
23  using namespace rapidxml;

```

Рисунок 17 – Подключение необходимых библиотек и объявление пространства имен RapidXML

Вывод обработанных данных в XML файлы включает в себя следующие шаги (рисунки 18, 19):

1. Создается пустой XML-документ.
2. В нем создается декларационный узел, в котором указаны такие атрибуты как версия стандарта XML, которой соответствует XML-документ, и имя кодировки, используемой при сохранении xml_document в файле или потоке.
3. Далее создается корневой узел, содержащий все последующие узлы. Атрибутами данного узла являются его имя, фрагмент, указывающий, что элементы и типы данных, используемые в схеме, взяты из указанного пространства имен, и фрагмент, указывающий на расположение, используемое для соответственного пространства имен.
4. Затем, в цикле, создается узел для графа, который будет содержать узлы для входящих в него вершин. Атрибутами узла для графа являются название

класса объекта, а также, полученные в ходе алгоритма обработки данных, характеристики графа.

5. Далее, также в цикле, создается узел для вершины, который будет содержать узел для соседей этой вершины. Параметрами узла для вершины являются название класса объекта, имя/номер вершины, координаты вершины и порядковый номер вершины в графе.

6. Затем создается узел для соседей текущей вершины. Атрибуты узла для соседей – название объекта класса, список соседей текущей вершины и список их порядковых номеров в текущем графе.

7. Далее шаги 5, 6 повторяются до тех пор, пока не будут перечислены все вершины текущего графа.

8. Далее шаги с 4 по 6 повторяются до тех пор, пока не будут перечислены все графы с их вершинами и соседями вершин.

9. Затем документ очищается, удаляются все узлы и очищается пул памяти.

```
338 files[w] = file;
339 rapidxml::xml_document<> doc; шаг 1
340 xml_node<>* decl = doc.allocate_node(node_declaration); шаг 2
341 decl->append_attribute(doc.allocate_attribute("version", "1.0"));
342 decl->append_attribute(doc.allocate_attribute("encoding", "utf-8"));
343 doc.append_node(decl);
344 xml_node<>* AOS = doc.allocate_node(node_element, "ArrayOfSimplex"); шаг 3
345 AOS->append_attribute(doc.allocate_attribute("xmlns:xsl", "http://www.w3.org/2001/XMLSchema-instance"));
346 AOS->append_attribute(doc.allocate_attribute("xmlns:xsd", "http://www.w3.org/2001/XMLSchema"));
347 doc.append_node(AOS);
348 for (int s = 0; s < files[w].size(); ++s)
349 {
350     xml_node<>* S = doc.allocate_node(node_element, "Simplex"); шаг 4
351     char cd0[30];
352     sprintf_s(cd0, "%g", files[w][s].get_gr_en_for_r0());
353     S->append_attribute(doc.allocate_attribute("E", doc.allocate_string(cd0)));
354     S->append_attribute(doc.allocate_attribute("Length", doc.allocate_string(to_string(files[w][s].get_gr_len()).c_str())));
355     S->append_attribute(doc.allocate_attribute("Diameter", doc.allocate_string(to_string(files[w][s].get_gr_diam()).c_str())));
356     S->append_attribute(doc.allocate_attribute("Radius", doc.allocate_string(to_string(files[w][s].get_gr_rad()).c_str())));
357     S->append_attribute(doc.allocate_attribute("Name", doc.allocate_string(files[w][s].gr_name.c_str())));
358     AOS->append_node(S);
359     for (int v = 0; v < files[w][s].vertices.size(); ++v)
360     {
361         xml_node<>* V = doc.allocate_node(node_element, "Vertex"); шаг 5
362         V->append_attribute(doc.allocate_attribute("Name", doc.allocate_string(files[w][s].vertices[v].vert_name.c_str())));
363         char cd1[14], cd2[14];
364         sprintf_s(cd1, "%g", files[w][s].vertices[v].y);
365         V->append_attribute(doc.allocate_attribute("y", doc.allocate_string(cd1)));
366         sprintf_s(cd2, "%g", files[w][s].vertices[v].x);
367         V->append_attribute(doc.allocate_attribute("x", doc.allocate_string(cd2)));
368         V->append_attribute(doc.allocate_attribute("Eo", ""));
369         V->append_attribute(doc.allocate_attribute("IndexInSimplex", doc.allocate_string(to_string(files[w][s].vertices[v].vert_num_in_gr).c_str())));
370         S->append_node(V);
371         if (!files[w][s].vertices[v].neighbors.empty())
372         {
373             xml_node<>* N = doc.allocate_node(node_element, "Neighbors"); шаг 6
374             std::ostringstream attr0, attr1;
375             std::copy(files[w][s].vertices[v].neighbors_names.begin(), files[w][s].vertices[v].neighbors_names.end() - 1, std::ostream_iterator<string>(attr0, " "));
376             attr0 << files[w][s].vertices[v].neighbors_names.back();
377             char catr0[64], catr1[64];
378             sprintf_s(catr0, "%s", attr0.str().c_str());
379             N->append_attribute(doc.allocate_attribute("ListOfNames", doc.allocate_string(catr0)));
380             std::copy(files[w][s].vertices[v].neighbors_nums_in_gr.begin(), files[w][s].vertices[v].neighbors_nums_in_gr.end() - 1, std::ostream_iterator<int>(attr1, " "));
381             attr1 << files[w][s].vertices[v].neighbors_nums_in_gr.back();
382             sprintf_s(catr1, "%s", attr1.str().c_str());
383             N->append_attribute(doc.allocate_attribute("ListOfIndex", doc.allocate_string(catr1)));
384             V->append_node(N);
385         }
386     }
```

Рисунок 18 – Реализация вывода обработанных данных в XML файл, шаги с

```

386     else
387     {
388         xml_node<>* N = doc.allocate_node(node_element, "Neighbors");
389         N->append_attribute(doc.allocate_attribute("ListOfNames", "-"));
390         N->append_attribute(doc.allocate_attribute("ListOfIndex", "-"));
391         V->append_node(N);
392     }
393 }
394 }
395 std::ofstream ftest(paths1[w]);
396 ftest << doc;
397 ftest.close();
398 doc.clear(); ————— шаг 9

```

Рисунок 19 – Реализация вывода обработанных данных в XML файл, шаг 9

3.1.5 Реализация распараллеливания

Подключение технологии MPI и реализация распараллеливания показаны на рисунках с 20 по 28.

microsoft.com/en-us/download/details.aspx?id=57467

Microsoft MPI v10.0

Important! Selecting a language below will dynamically change the complete page content to that language.

Language: English Download

Stand-alone, redistributable and SDK installers for Microsoft MPI

⊖ Details

Note: There are multiple files available for this download. Once you click on the "Download" button, you will be prompted to select the files you need.

| | |
|---|---------------------------------------|
| Version: 10.0.12498.5 | Date Published: 4/29/2021 |
| File Name: msmpisetup.exe msmpisdk.msi | File Size: 7.5 MB 2.4 MB |

Рисунок 20 – Скачивание файлов MPI

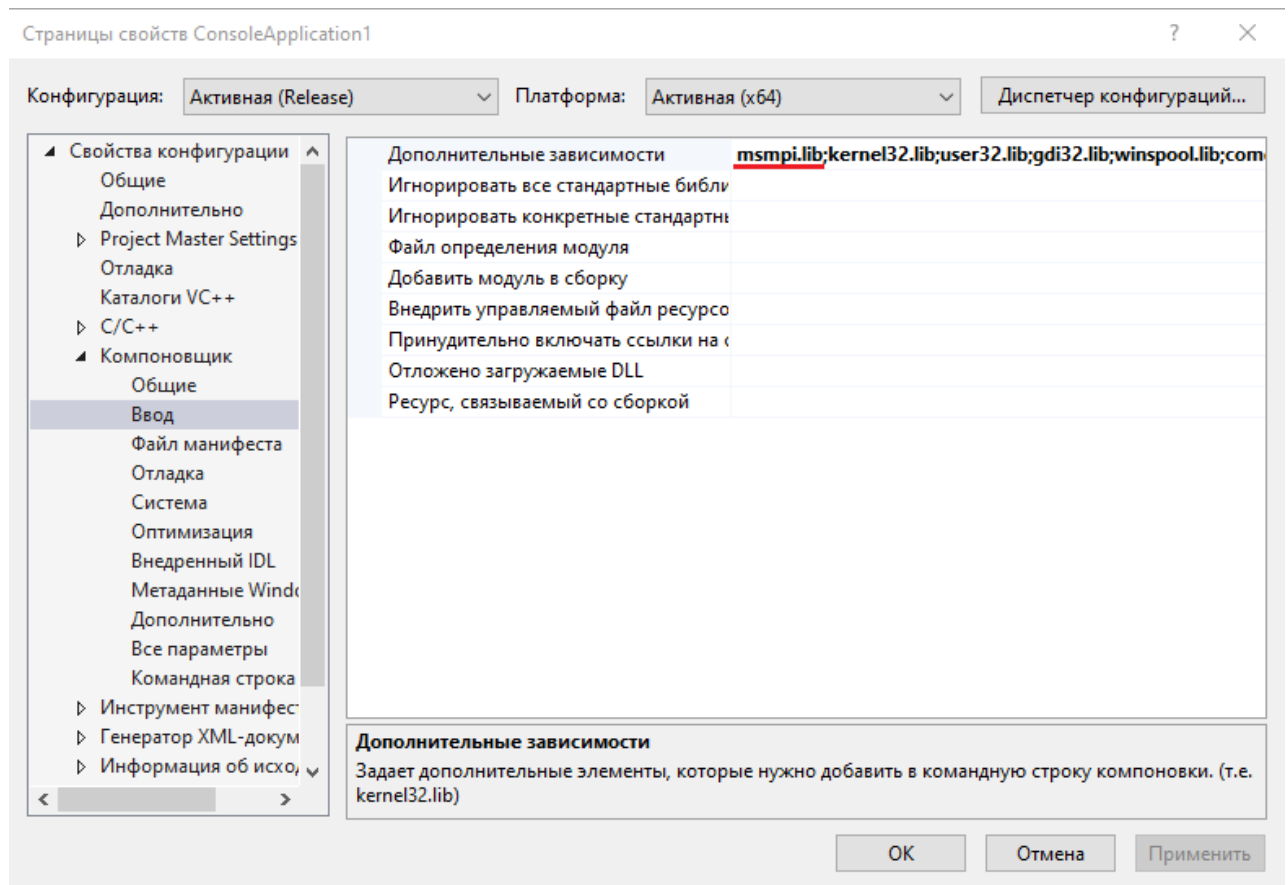


Рисунок 21 – Добавление зависимости msmpi.lib

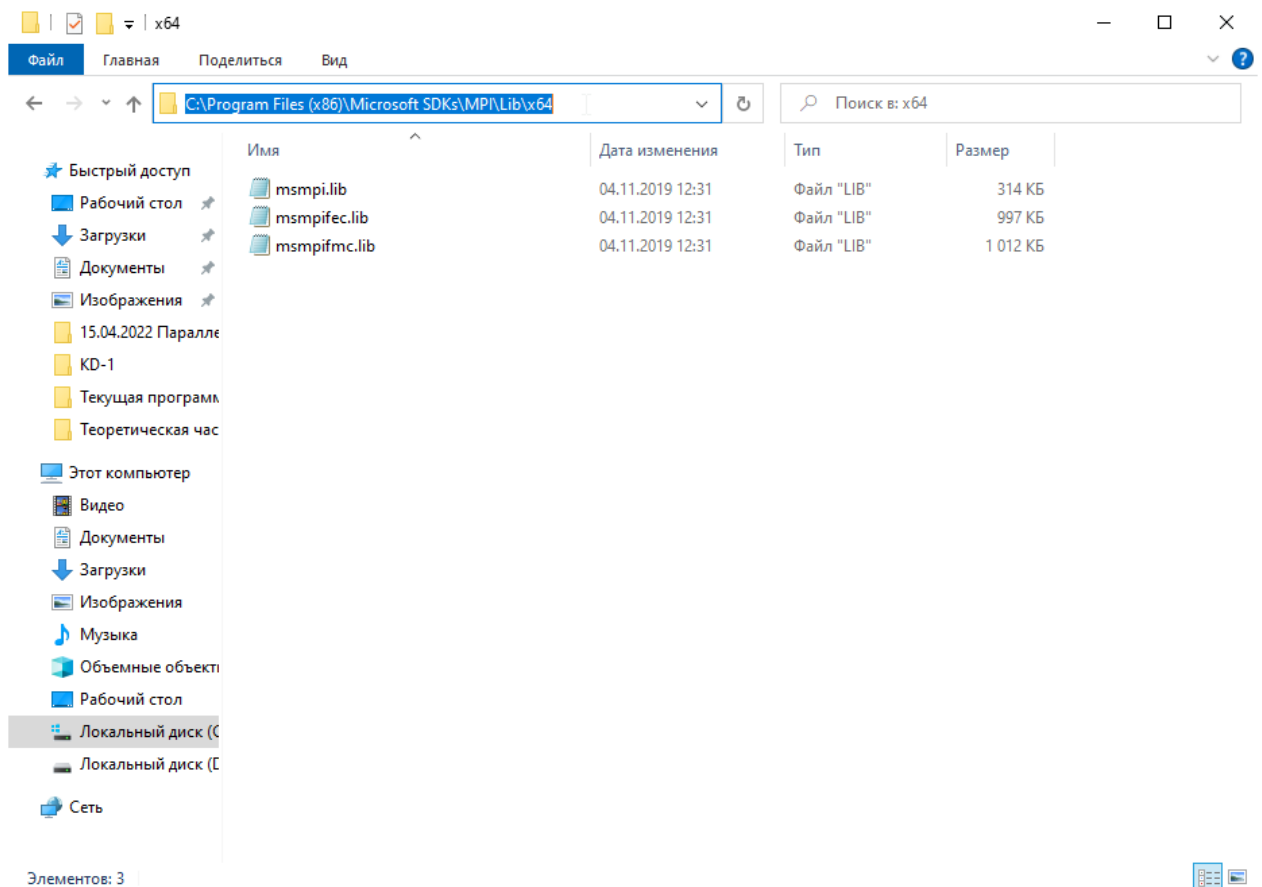


Рисунок 22 – Путь до папки x64

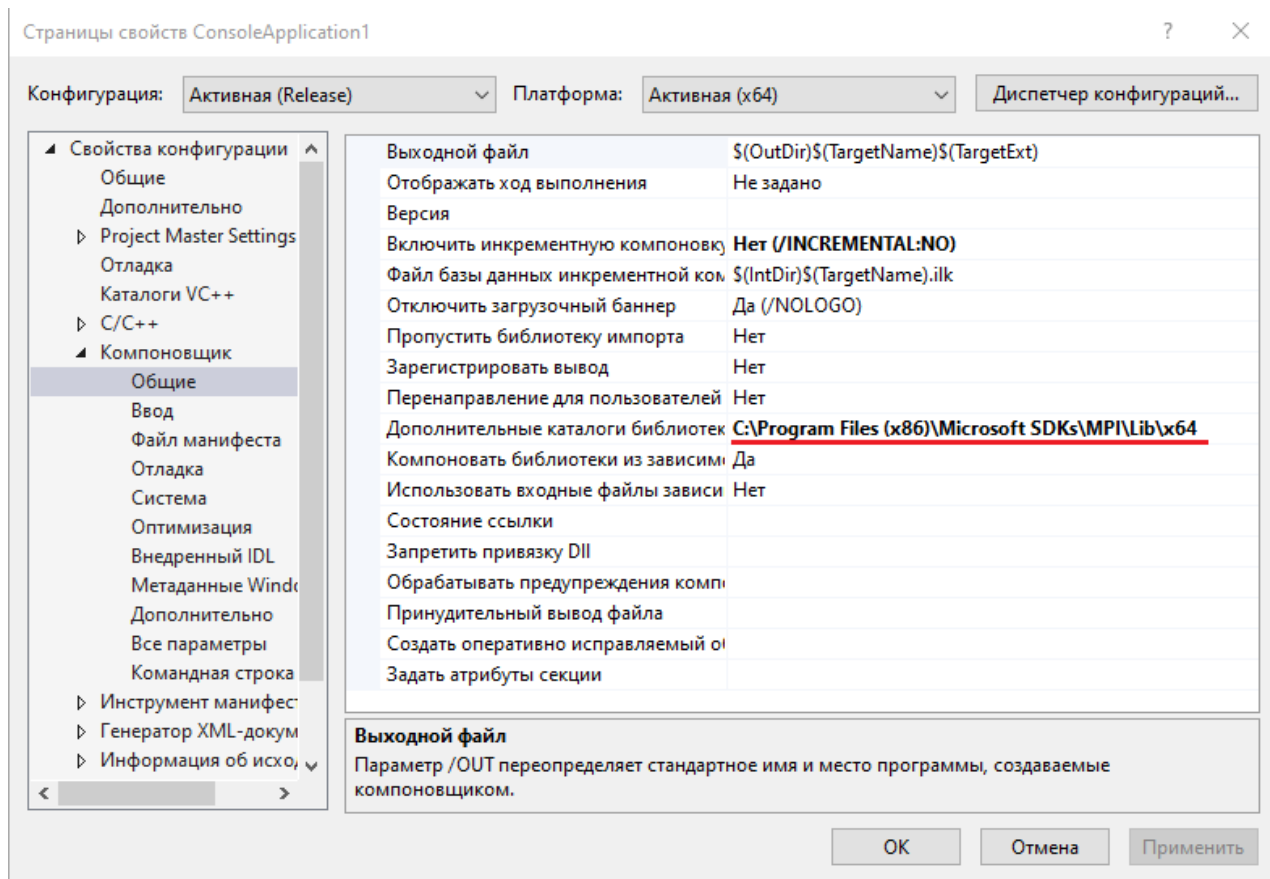


Рисунок 23 – Добавление каталога библиотеки MPI

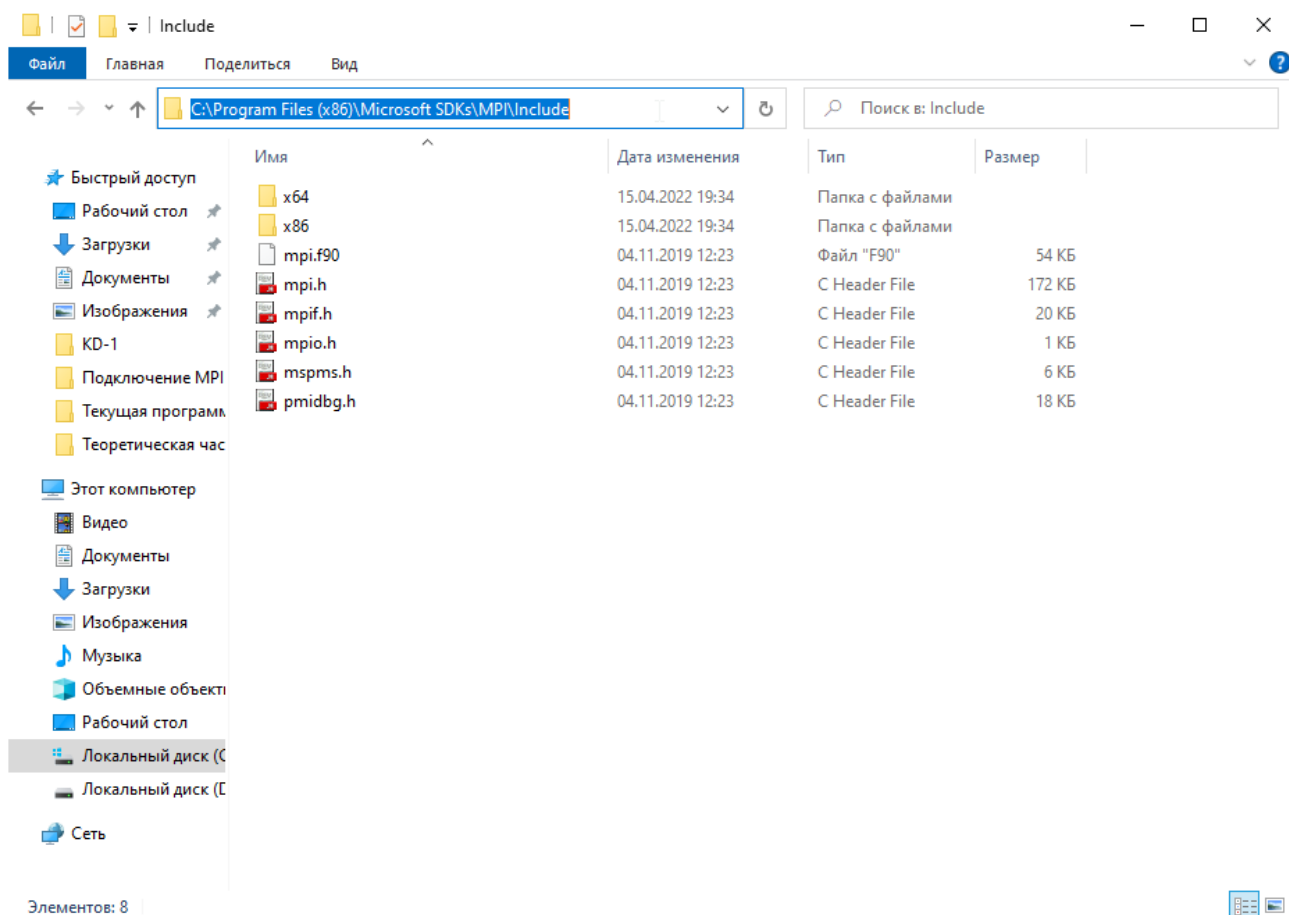


Рисунок 24 – Путь до папки include

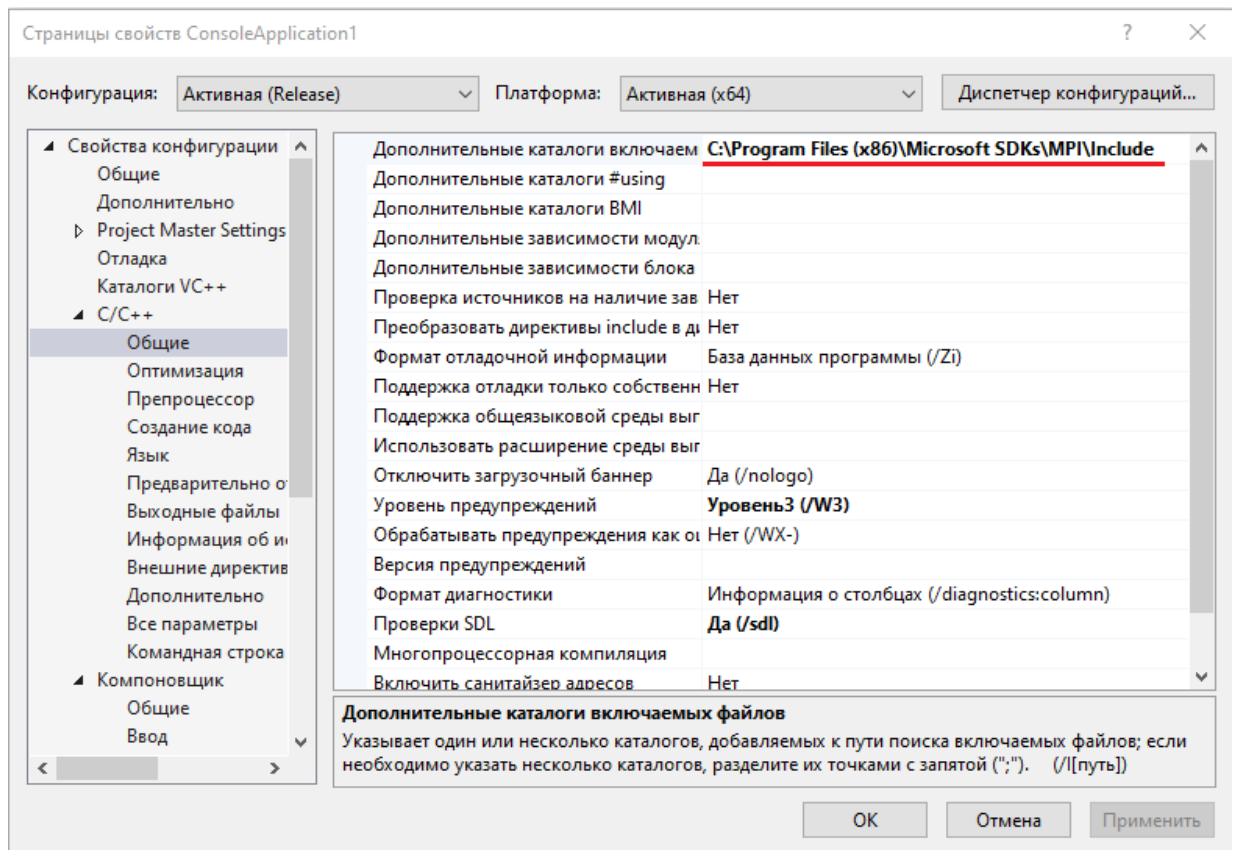


Рисунок 25 – Добавление дополнительного каталога MPI

```
19 | #include "mpi.h"
```

Рисунок 26 – Подключение библиотеки mpi.h

```
203 | MPI_Request* reqs = new MPI_Request[2 * input_file_names.size() - 1];
204 | MPI_Status* stats = new MPI_Status[2 * input_file_names.size() - 1];
205 | MPI_Init(&argc, &argv);
206 | MPI_Comm_rank(MPI_COMM_WORLD, &rank);
207 | double start = MPI_Wtime(), wtick = MPI_Wtick();
208 | if (rank == 0)
209 | {
210 |     cout << "number of txt files = " << 1 << endl;
211 |     w = 0;
212 | }
213 | else
214 | {
215 |     w = rank;
216 | }
```

Рисунок 27 – Начало работы с MPI, инициализация параллельной области

```

423     if (rank == 0)
424     {
425         ends[0] = MPI_Wtime();
426         #pragma omp parallel for num_threads(input_file_names.size())
427         for (int rank1 = 1; rank1 < input_file_names.size(); ++rank1)
428         {
429             MPI_Irecv(&ends[rank1], 1, MPI_DOUBLE, rank1, 0, MPI_COMM_WORLD, &reqs[rank1 - 1]);
430         }
431         MPI_Waitall(2 * input_file_names.size() - 2, reqs, stats);
432         int i;
433         auto result = max_element(ends.begin(), ends.end());
434         i = std::distance(ends.begin(), result);
435         printf_s("Program execution time = %g\n", ends[i] - start);
436         printf_s("System timer accuracy = %f", wtick);
437     }
438     else
439     {
440         ends[rank] = MPI_Wtime();
441         MPI_Isend(&ends[rank], 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &reqs[input_file_names.size() - 1 +
442     }
443     delete[] reqs;
444     delete[] stats;
445     MPI_Finalize();

```

Рисунок 28 – Конец работы с MPI, завершение параллельной области

3.2 Преимущества разработанной программы

Модифицированный алгоритм поиска в ширину, реализованный в данной программе, уникален тем, что находит вершины, принадлежащие графу, и соседей данной вершины с помощью определения расстояний между вершинами и ограничения на допустимое расстояние между ними.

Еще одним достоинством алгоритма является применение объектно-ориентированного подхода к его созданию, то есть возможность создавать объекты и присваивать им характеристики в процессе обработки данных.

В приложении используется самая быстрая из ныне существующих технологий параллельного программирования MPI, позволяющая ускорить и так быстрый и максимально оптимизированный алгоритм обработки данных почти в 4 раза (представлено в таблице 1 результатов тестирования).

Также, неоспоримым преимуществом разработанного приложения является использование самого быстрого на сегодняшний день XML анализатора (парсера) RapidXML для вывода обработанных данных в XML файлы [31]. Его скорость в десятки раз превышает скорость многих других XML анализаторов.

3.3 Тестирование программы

Во время разработки программы было создано 2 версии, последовательная и параллельная, с применением технологии MPI. Обе версии проекта тестировались в двух режимах: Debug и Release. Тестирование проводилось на текстовых файлах, находящихся в одном хранилище. Разработанная программа не рассчитана на то, что файлы будут находиться на разных носителях или серверах. Версия MPI запускалась в командной строке cmd с помощью команды «mpirun -n m ConsoleApplication1» (при использовании команды скобки убрать), где mpirun – команда запуска исполняющего exe файла, m – количество процессов, которое нужно породить (для данной программы “m” обязательно нужно прописывать, иначе в результате на выходе получится только один заполненный xml файл), ConsoleApplication1 – название готового приложения. В таблице 1 представлены результаты тестирования обеих версий.

Таблица 1 – Результаты тестирования последовательной и параллельной версий программы в Debug и Release режимах

| Версия | Режим | Тест 1, с | Тест 2, с | Тест 3, с | Тест 4, с | Тест 5, с | Среднее, с |
|------------------|---------|-----------|-----------|-----------|-----------|-----------|------------|
| Последовательная | Debug | 6,31363 | 6,26594 | 6,17236 | 6,2005 | 6,14894 | 6,220274 |
| | Release | 2,37825 | 2,37969 | 2,39 | 2,3746 | 2,39669 | 2,383846 |
| Параллельная | Debug | 1,66042 | 1,56523 | 1,53667 | 1,54758 | 1,55429 | 1,57474 |
| | Release | 0,59558 | 0,596234 | 0,594725 | 0,646744 | 0,602159 | 0,6070884 |

Исходя из результатов, лучшее среднее время выполнения показала параллельная версия программы в Release-режиме. Она быстрее последовательной версии в Release-режиме почти в 4 раза (~3,927...).

В таблице 2 представлены результаты тестирования обеих версий в Release-режиме, показывающие влияние количества тестируемых файлов на скорость выполнения программы.

Таблица 2 – Результаты тестирования обеих версий программы в Release-режиме на разном количестве входных файлов для демонстрации влияния количества тестируемых файлов на скорость выполнения программы

| Версия | Тест 1 - 4 файла, с | Тест 2 - 8 файлов, с | Тест 3 - 12 файлов, с | Тест 4 - 16 файлов, с |
|------------------|---------------------|----------------------|-----------------------|-----------------------|
| Последовательная | 2,37837 | 4,76813 | 7,13412 | 9,49064 |
| Параллельная | 0,693566 | 0,831057 | 0,948171 | 1,34249 |

Итого, по результатам второго теста видно, что количество файлов влияет обратно пропорционально на скорость выполнения программы.

В таблице 3 представлены результаты тестирования параллельной версии программы в Release-режиме, показывающие влияние количества тестируемых файлов на объем необходимой памяти для вычислений.

Таблица 3 – Результаты тестирования параллельной версии программы в Release-режиме на разном количестве входных файлов для демонстрации влияния количества тестируемых файлов на объем необходимой памяти для вычислений

| Режим | Тест 1 - 4 файла, байты | Тест 2 - 8 файлов, байты | Тест 3 - 12 файлов, байты | Тест 4 - 16 файлов, байты |
|---------|----------------------------|-----------------------------|------------------------------|------------------------------|
| Debug | 73440 | 149440 | 228000 | 307840 |
| Release | 58752 | 119552 | 182400 | 246272 |

Исходя из результатов третьего теста, видно, что, чем больше файлов нужно обработать, тем больше памяти требуется для вычислений. Также, по таблице видно, что при работе с одинаковым количеством входных файлов в обоих режимах, в Release-режиме каждый раз на вычисления затрачивается в 1,25 раз меньше памяти.

Подводя итог, очевидно, что самым эффективным и экономным с точки зрения времени и памяти, необходимой для вычислений, вариантом программы является параллельная версия в Release-режиме.

Заключение

В ходе данной работы была разработана программа для параллельной обработки изображений.

В ходе работы были получены теоретические знания в области параллельного программирования и анализа xml документов. Также, были получены практические навыки работы с технологиями параллельных вычислений OpenMP и MPI, навыки работы с xml-анализаторами, такими как TinyXML2, Pugixml и RapidXML.

Были решены следующие задачи:

- реализован класс «граф»;
- реализован класс «вершина»;
- реализован класс «сосед»;
- разработан алгоритм для обработки входных данных;
- реализован вывод упорядоченных объектов, полученных из входных данных, в файлы с расширением .xml.
- реализовано параллельное чтение файлов, обработка данных и вывод в файлы.

В результате проведенного тестирования была определена самая быстрая версия приложения, а именно параллельная версия в Release-режиме. Данная программа почти в 4 раза быстрее обычной последовательной версии в таком же режиме. Параллельная версия отлично подойдет для обработки очень большого количества данных и может быть использована для разных задач, связанных с обработкой изображений.

Практическая значимость работы заключается в создании программы с реализованным в ней алгоритмом, который уникален тем, что позволяет считывать данные из файлов вместо стандартного использования матрицы смежности в качестве входных данных, а также находить вершины графов за счет определения расстояний между вершинами. Помимо этого, алгоритм в ходе

обработки данных вычисляет параметры созданных объектов для последующего их вывода в xml файлы. Еще одной отличительной особенностью программы является то, что все 3 основных этапа, а именно считывание данных, их обработка и вывод, выполняются параллельно в четырех разных процессах. Использование самых быстрых технологии параллельных вычислений и xml парсера положительно сказываются на скорости обработки большого количества данных и выводе их в xml файлы соответственно, а, следовательно, и на значительной экономии времени, когда нужно обработать изображения.

Дальнейшее совершенствование алгоритма может происходить за счет добавления новых методов в классы либо замены существующих, либо, в крайнем случае, добавления новых классов. В остальном, приложение максимально производительное, так как оптимизировано и использует самые быстрые на сегодняшний день технологию параллельных вычислений MPI и XML-анализатор для вывода данных RapidXML.

Список литературы

1. Seyed H. Roosta Parallel Processing and Parallel Algorithms: Theory and Computation. – Springer, 1999. – 585 p.
2. hyPACK-2013 Four Days Technology Workshop [Электронный ресурс]. Mode 1: POSIX Thread Programming (Pthreads). Режим доступа: <http://www.tezu.ernet.in/dcompsc/facility/HPCC/hypack/pthreads-hypack-2013/pthreads-overview.html> (дата обращения 01.04.2022).
3. Ayon Basumallik, Seung-Jai Min, Rudolf Eigenmann Programming Distributed Memory Sytems Using OpenMP. - IEEE International Parallel and Distributed Processing Symposium, 2007. – 8 p.
4. Abraham Silberschatz, Peter B. Galvin, Gerg Gagne Operating System Concepts. - 9ed, ISV. – John Wiley & Sons, 2008. – 919 p.
5. J.J. Costa, T. Cortes, X. Martorell, E. Ayguade, J. Labarta Running OpenMP applications efficiently on an everything-shared SDSM. – Journal of Parallel and Distributed Computing, 2006. – 27 p.
6. Wikipedia [Электронный ресурс]. OpenMP. Режим доступа: <https://en.wikipedia.org/wiki/OpenMP> (дата обращения 01.04.2022).
7. Wikipedia [Электронный ресурс]. Parallel Virtual Machine. Режим доступа: https://en.wikipedia.org/wiki/Parallel_Virtual_Machine (дата обращения 01.04.2022).
8. Wikipedia [Электронный ресурс]. Unified Parallel C. Режим доступа: https://en.wikipedia.org/wiki/Unified_Parallel_C (дата обращения 10.04.2022).
9. Gilberto Contreras, Margaret Martonosi Characterizing and Improving the Performance of Intel Threading Building Blocks. - Department of Electrical Engineering Princeton University, 2008. – 10 p.
10. Robert L. Bocchino Jr., Vikram S. Adve, Sarita V. Adve and Marc Snir Parallel Programming Must Be Deterministic by Default. - University of Illinois at Urbana-Champaign, 2009. – 6 p.

11. Robert Demming, Daniel J. Duffy Introduction to the Boost C++ Libraries Volume I - Foundations. - Datasim Education BV, 2010. – 316 p.
12. Robert Demming, Daniel J. Duffy Introduction to the Boost C++ Libraries Volume II – Advanced Libraries. - Datasim Education BV, 2012. – 354 p.
13. R. C. Ward Engineering physics and mathematics division progress report for period ending December 31, 1992. – ORNL-6753, 1993. – 273 p.
14. CORPORATE The MPI Forum MPI: a message passing interface. - Supercomputing '93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing, 1993. – 6 p.
15. Geeksforgeeks [Электронный ресурс]. OOPs | Object Oriented Design. Режим доступа: <https://www.geeksforgeeks.org/oops-object-oriented-design/> (дата обращения 06.05.2022).
16. Collegenote [Электронный ресурс]. Object Oriented Programming Approach. Режим доступа: <https://www.collegenote.net/curriculum/object-oriented-programming/67/358/> (дата обращения 06.05.2022).
17. Geeksforgeeks [Электронный ресурс]. Differences between Procedural and Object Oriented Programming. Режим доступа: <https://www.geeksforgeeks.org/differences-between-procedural-and-object-oriented-programming/> (дата обращения 06.05.2022).
18. Even, Shimon Graph Algorithms. – 2nd ed. - Cambridge University Press, 2011. – 2 p.
19. Geeksforgeeks [Электронный ресурс]. Depth First Search or DFS for a Graph. Режим доступа: <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/> (дата обращения 10.04.2022).
20. Reif, John H. Depth-first search is inherently sequential. - Information Processing Letters, 1985. – 5 p.
21. A. Aggarwal, Anderson, R. J. A random NC algorithm for depth first search. – Combinatorica, 1988. – 11 p.
22. Konrad Zuse Der Plankalkül. - Internet Archive, 1972. – 180 p.

23. Hackerearth [Электронный ресурс]. Breadth First Search. Режим доступа: <https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/> (дата обращения 26.04.2022).
24. Edward F. Moore The Shortest Path Through a Maze. - Harvard University Press, 1959. – 8 p.
25. Steven S. Skiena The Algorithm Design Manual. – Springer, 2008. – 730 p.
26. Medium [Электронный ресурс]. Advantages and Disadvantages of Search Algorithms. Режим доступа: <https://www.vtupulse.com/artificial-intelligence/breadth-first-search-example-advantages-and-disadvantages/> (дата обращения 04.05.2022).
27. Grinninglizard [Электронный ресурс]. TinyXml Documentation. Режим доступа: <http://www.grinninglizard.com/tinyxmldocs/> (дата обращения 04.05.2022).
28. Wikipedia [Электронный ресурс]. Libxml2. Режим доступа: <https://en.wikipedia.org/wiki/Libxml2> (дата обращения 04.05.2022).
29. Gitlab.gnome [Электронный ресурс]. Libxml2. Режим доступа: <https://gitlab.gnome.org/GNOME/libxml2/-/wikis/home> (дата обращения 04.05.2022).
30. Pugixml [Электронный ресурс]. Pugixml. Режим доступа: <https://pugixml.org/> (дата обращения 04.05.2022).
31. Rapidxml.sourceforge [Электронный ресурс]. RAPIDXML Manual. Режим доступа: <http://rapidxml.sourceforge.net/manual.html> (дата обращения 04.05.2022).
32. Dartmouth [Электронный ресурс]. Pros and Cons of OpenMP/MPI. Режим доступа: https://www.dartmouth.edu/~rc/classes/intro_mpi/parallel_prog_compare.html (дата обращения 10.04.2022).

Приложение А

Код программы

```
#include <stdio.h>
#include <iostream>
#include <queue>
#include <string>
#include <list>
#include <fstream>
#include <string>
#include <cmath>
#include <vector>
#include <algorithm>
#include <sstream>
#include <cstdlib>
#include <rapidxml/rapidxml.hpp>
#include <rapidxml/rapidxml_print.hpp>
#include <rapidxml/rapidxml_iterators.hpp>
#include <rapidxml/rapidxml_utils.hpp>
#include <iomanip>
#include <windows.h>
#include "mpi.h"
#include <filesystem>
#include <boost/filesystem.hpp>
#include <locale.h>
#include <omp.h>

using std::string;
using namespace std;
using namespace rapidxml;
namespace fs = std::filesystem;

class Graph
{
```

```

public:
    Graph()
    {

    }
    class Vertex
    {
    public:
        Vertex()
        {

        }
        class Neighbor
        {
        public:
            Neighbor()
            {

            }
            string neighbor_name;
            int neighbor_num_in_gr;
            ~Neighbor()
            {

            }
        };
        string vert_name;
        int vert_num_in_gr;
        double x, y;
        vector<string> neighbors_names;
        vector<int> neighbors_nums_in_gr;
        vector<Neighbor> neighbors;
        size_t      get_am_of_neighbors_of_vert()//Vertex
метод: Получить число соседей
    {

```

```

        return neighbors.size();
    }
    void      add_neighbor_in_neighbors_list(Neighbor
neighbor)//Vertex метод: Добавить соседа
    {
        neighbors.push_back(neighbor);
    }
    Neighbor      get_neighbor(int
neighbor_num_in_vert)//Vertex метод: Получить объект сосед по
номеру в списке
    {
        return neighbors[neighbor_num_in_vert];
    }
    ~Vertex()
    {

    }

};
string gr_name;
double gr_energy;
vector<Vertex> vertices;
int      gr_radius,      gr_diameter,      gr_length,
gr_amount_of_edges;
vector<string> vertices_list;
vector<double> distances;
size_t  get_am_of_vert_in_gr();//Graph метод: Получить
число вершин в графе
    {
        return vertices.size();
    }
    Vertex  get_vert(int  vert_num_in_gr)//Graph метод:
Получить объект вершина по номеру вершины в графе
    {
        return vertices[vert_num_in_gr];
    }

```

```

        int get_gr_rad()//Graph метод: Получить/вычислить
радиус графа
    {
        if (distances.size() > 0)
        {
            int i;
            auto result1 =
min_element(distances.begin(), distances.end());
            i = std::distance(distances.begin(),
result1);
            gr_radius = distances[i];
        }
        else
        {
            gr_radius = 0;
        }
        return gr_radius;
    }
    int get_gr_diam()//Graph метод: Получить/вычислить
диаметр графа
    {
        if (distances.size() > 0)
        {
            int i;
            auto result2 =
max_element(distances.begin(), distances.end());
            i = std::distance(distances.begin(),
result2);
            gr_diameter = distances[i];
        }
        else
        {
            gr_diameter = 0;
        }
        return gr_diameter;
    }

```

```

    }
    int    get_gr_len()//Graph    метод:    Получить/вычислить
длину графа
    {
        gr_length = gr_amount_of_edges;
        return gr_length;
    }
    vector<string>    get_string_vert_list()//Graph    метод:
Получить список вершин в текстовом формате
    {
        for (int i = 0; i < vertices.size(); ++i)
        {

vertices_list.push_back(vertices[i].vert_name);
        }
        return vertices_list;
    }
    double    get_gr_en_for_r0()//Graph    метод:
Получить/вычислить энергию графа для заданного R0
    {
        if (distances.size() > 0)
        {
            gr_energy = 0;
            double r0 = 10.0;
            for (int i = 0; i < distances.size(); ++i)
            {
                gr_energy += pow((r0/distances[i]), 12)
- pow((r0 / distances[i]), 6);
            }
            return gr_energy;
        }
        else
        {
            gr_energy = 0;
        }
    }

```

```

        return gr_energy;
    }

    void add_vert_in_vert_list(Vertex vertex) //Graph
метод: Добавить объект вершина в список вершин графа
    {
        vertices.push_back(vertex);
    }
    ~Graph()
    {

    }
};

int main(int argc, char *argv[])
{
    setlocale(LC_ALL, "Russian");
    string path0 = "D:\\Дисциплины Семестр
8\\ВКР\\17.06.2022 Параллельная программа MPI для любого
количества файлов\\Входные файлы\\"; //Поменять путь к папке с
входными файлами на свой
    string path1;
    vector<string> input_file_names;
    vector<string> output_file_names;
    int l = 0;
    for (const auto& entry :
fs::directory_iterator(path0))
    {
        path1 = entry.path().string();
        if (path1.substr(path1.find_last_of(".") + 1) ==
"txt")
        {
            input_file_names.push_back(path1);
            //cout << input_file_names[l] << "\n";
            boost::filesystem::path p = path1;

```



```

        output_file_names.push_back("D:\\Дисциплины
Семестр 8\\ВКР\\17.06.2022 Параллельная программа MPI для
любого количества файлов\\Выходные файлы\\" + p.stem().string()
+ ".xml"); //Поменять путь к папке для выходных файлов на свой
        //cout << output_file_names[l] << "\n\n";
        ++l;
    }
}
if (l == 0)
{
    cout << "No txt files" << endl;
}
//cout << "number of txt files = " << i << endl;
vector<vector<Graph>> files(input_file_names.size());
int rank, w, i = 0, it = 0, it1 = 0, c0, c1, c2 = 0,
it2;

vector<double> ends(input_file_names.size()), x, y;
vector<string> vn;
vector<Graph> file;
string Vnum, rubbish1, Xm, Ym, rubbish2, path;
vector<int> nodes, nodes1, numnodes;
std::vector<int>::iterator it4;
ifstream fin;
//MPI_Request reqs[6];
//MPI_Status stats[6];
MPI_Request* reqs = new MPI_Request[2 *
input_file_names.size() - 1];
MPI_Status* stats = new MPI_Status[2 *
input_file_names.size() - 1];
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
double start = MPI_Wtime(), wtick = MPI_Wtick();
if (rank == 0)
{
    cout << "number of txt files = " << l << endl;

```

```

        w = 0;
    }
else
{
    w = rank;
}
path = input_file_names[w];
fin.open(path);
if (!fin.is_open())
{
    cout << "File opening error!" << "\n";
}
else
{
    //cout << "File open!" << "\n";
    string str;
    getline(fin, str);
    while (!fin.eof())
    {
        getline(fin, Vnum, '\t');
        vn.push_back(Vnum);
        getline(fin, rubbish1, '\t');
        getline(fin, Xm, '\t');
        x.push_back(stof(Xm));
        getline(fin, Ym, '\t');
        y.push_back(stof(Ym));
        getline(fin, rubbish2, '\n');
        ++i;
    }
    fin.close();
}
for (int j = 0; j < i; ++j)
{
    nodes.push_back(0);
    nodes1.push_back(0);
}

```

```

        numnodes.push_back(0);
    }
    queue<int> Queue, Queue1;
    Queue1.push(0);
    while (!Queue1.empty())
    {
        if (it == 0)
        {
            int n = Queue1.front();
            Queue1.pop();
            Queue.push(n);
        }
        if (it == 1)
        {
            int n, h;
            for (int p = 0; p < i; ++p)
            {
                n = Queue1.front();
                h = numnodes[p];
                if (n == h)
                {
                    n = 0;
                    Queue1.pop();
                }
            }
            if (!Queue1.empty())
            {
                Queue1.pop();
            }
            Queue.push(n);
        }
        it = 1;
        if (!Queue.empty())
        {
            Graph graph;

```

```

vector<vector<int>> verts_of_gr;
graph.gr_name = to_string(++c2);
int node;
c0 = 0;
c1 = 0;
it2 = 0;
while (!Queue.empty())
{
    Graph::Vertex vertex;
    node = Queue.front();
    Queue.pop();
    nodes[node] = 2;
    vertex.vert_name = vn[node];
    vertex.vert_num_in_gr = c0;
    vertex.x = x[node];
    vertex.y = y[node];
    nodes1[node] = 3;
    for (int j = 0; j < i; ++j)
    {
        double d = sqrt(pow((x[j] -
x[node]), 2) + pow((y[j] - y[node]), 2));
        if (9 <= d && d <= 11)
        {
            Graph::Vertex::Neighbor
neighbor;

            if (nodes[j] == 0)
            {
                ++c1;
                Queue.push(j);
                nodes[j] = 1;
                numnodes[j] = j;
                nodes1[j] = 3;
            }
            graph.distances.push_back(d);

```

```

neighbor.neighbor_name =
vn[j];

vertex.neighbors_names.push_back(vn[j]);

vertex.add_neighbor_in_neighbors_list(neighbor);
    }
    }
    graph.add_vert_in_vert_list(vertex);
    ++c0;
    }
    for (int i = 0; i < graph.vertices.size();
++i)
    {
        for (int j = 0; j <
graph.vertices[i].neighbors.size(); ++j)
        {
            for (int z = 0; z <
graph.vertices.size(); ++z)
            {
                if
(graph.vertices[i].neighbors[j].neighbor_name ==
graph.vertices[z].vert_name)
                {

                    graph.vertices[i].neighbors[j].neighbor_num_in_gr =
graph.vertices[z].vert_num_in_gr;

                    graph.vertices[i].neighbors_nums_in_gr.push_back(graph.vert
ices[z].vert_num_in_gr);

                }
            }
        }
    }
    if (it1 == 0)

```

```

        {
            for (int j = 0; j < i; ++j)
            {
                if (nodes[j] == 0)
                {
                    Queue1.push(j);
                    nodes1[j] = 3;
                }
            }
        }
        graph.gr_amount_of_edges = c1;
        file.push_back(graph);
        it1 = 1;
        it2 = 1;
    }
    int f0 = 0, f1 = 1;
    it4 = std::find(nodes.begin(), nodes.end(), f0);
    if (it4 != nodes.end())
    {
        }
    else
    {
        break;
    }
}
files[w] = file;
rapidxml::xml_document<> doc;
xml_node<>* decl =
doc.allocate_node(node_declaration);
    decl-
>append_attribute(doc.allocate_attribute("version", "1.0"));
    decl-
>append_attribute(doc.allocate_attribute("encoding", "utf-8"));
    doc.append_node(decl);

```

```

        xml_node<>* AOS = doc.allocate_node(node_element,
"ArrayOfSimplex");
        AOS-
>append_attribute(doc.allocate_attribute("xmlns:xsi",
"http://www.w3.org/2001/XMLSchema-instance"));
        AOS-
>append_attribute(doc.allocate_attribute("xmlns:xsd",
"http://www.w3.org/2001/XMLSchema"));
        doc.append_node(AOS);
        for (int s = 0; s < files[w].size(); ++s)
        {
            xml_node<>* S = doc.allocate_node(node_element,
"Simplex");
            char cd0[30];
            sprintf_s(cd0, "%g",
files[w][s].get_gr_en_for_r0());
            S->append_attribute(doc.allocate_attribute("E",
doc.allocate_string(cd0)));
            S-
>append_attribute(doc.allocate_attribute("Length",
doc.allocate_string(to_string(files[w][s].get_gr_len()).c_str()
)));
            S-
>append_attribute(doc.allocate_attribute("Diameter",
doc.allocate_string(to_string(files[w][s].get_gr_diam()).c_str(
))));
            S-
>append_attribute(doc.allocate_attribute("Radius",
doc.allocate_string(to_string(files[w][s].get_gr_rad()).c_str()
)));
            S-
>append_attribute(doc.allocate_attribute("Name",
doc.allocate_string(files[w][s].gr_name.c_str())));
            AOS->append_node(S);

```

```

        for (int v = 0; v < files[w][s].vertices.size();
++v)
    {
        xml_node<>* V =
doc.allocate_node(node_element, "Vertex");
        V-
>append_attribute(doc.allocate_attribute("Name",
doc.allocate_string(files[w][s].vertices[v].vert_name.c_str()))
);
        char cd1[14], cd2[14];
        sprintf_s(cd1, "%g",
files[w][s].vertices[v].y);
        V-
>append_attribute(doc.allocate_attribute("y",
doc.allocate_string(cd1)));
        sprintf_s(cd2, "%g",
files[w][s].vertices[v].x);
        V-
>append_attribute(doc.allocate_attribute("x",
doc.allocate_string(cd2)));
        V-
>append_attribute(doc.allocate_attribute("Ec", ""));
        V-
>append_attribute(doc.allocate_attribute("IndexInSimplex",
doc.allocate_string(to_string(files[w][s].vertices[v].vert_num_
in_gr).c_str())));
        S->append_node(V);
        if
(!files[w][s].vertices[v].neighbors.empty())
        {
            xml_node<>* N =
doc.allocate_node(node_element, "Neighbors");
            std::ostringstream attr0, attr1;

            std::copy(files[w][s].vertices[v].neighbors_names.begin(),

```



```

files[w][s].vertices[v].neighbors_names.end() - 1,
std::ostream_iterator<string>(attr0, "; "));
    attr0 <<
files[w][s].vertices[v].neighbors_names.back();
    char cattr0[64], cattr1[64];
    sprintf_s(cattr0, "%s",
attr0.str().c_str());
    N-
>append_attribute(doc.allocate_attribute("ListOfNames",
doc.allocate_string(cattr0)));

    std::copy(files[w][s].vertices[v].neighbors_nums_in_gr.begi
n(), files[w][s].vertices[v].neighbors_nums_in_gr.end() - 1,
std::ostream_iterator<int>(attr1, "; "));
    attr1 <<
files[w][s].vertices[v].neighbors_nums_in_gr.back();
    sprintf_s(cattr1, "%s",
attr1.str().c_str());
    N-
>append_attribute(doc.allocate_attribute("ListOfIndex",
doc.allocate_string(cattr1)));
    V->append_node(N);
}
else
{
    xml_node<>* N =
doc.allocate_node(node_element, "Neighbors");
    N-
>append_attribute(doc.allocate_attribute("ListOfNames", "-"));
    N-
>append_attribute(doc.allocate_attribute("ListOfIndex", "-"));
    V->append_node(N);
}
}
}

```

```

std::ofstream ftest(output_file_names[w]);
ftest << doc;
ftest.close();
doc.clear();
if (rank == 0)
{
    ends[0] = MPI_Wtime();
    #pragma omp parallel for
num_threads(input_file_names.size())
    for (int rank1 = 1; rank1 <
input_file_names.size(); ++rank1)
    {
        MPI_Irecv(&ends[rank1], 1, MPI_DOUBLE,
rank1, 0, MPI_COMM_WORLD, &reqs[rank1 - 1]);
    }
    MPI_Waitall(2 * input_file_names.size() - 2,
reqs, stats);
    int i;
    auto result = max_element(ends.begin(),
ends.end());
    i = std::distance(ends.begin(), result);
    printf_s("Program execution time = %g\n", ends[i]
- start);
    printf_s("System timer accuracy = %f", wtick);
}
else
{
    ends[rank] = MPI_Wtime();
    MPI_Isend(&ends[rank], 1, MPI_DOUBLE, 0, 0,
MPI_COMM_WORLD, &reqs[input_file_names.size() - 1 + rank]);
}
delete[] reqs;
delete[] stats;
MPI_Finalize();
cin.get();

```

```
        return 0;  
    }
```

ОТЗЫВ
руководителя ВКР на работу обучающегося

Никитина Даниила Олеговича
Направление подготовки 09.03.02 «Информационные системы и технологии»
профиль Информационные системы и технологии

над выпускной квалификационной работой бакалавра
«Разработка и реализация алгоритмов параллельной обработки изображений для суперкомпьютера»

Актуальность работы

Обработка изображений является ресурсоемкой задачей, обработка сотен изображений большого объема требует значительных вычислительных мощностей и применения алгоритмов параллельной обработки. Изображения атомной структуры аморфных материалов трудно поддаются классификации и последующему анализу, выделение на изображениях упорядоченных атомных кластеров и их представление в виде графов позволит получить новую информацию о структуре объекта. Разработка приложения с параллельной обработкой координат атомов является актуальной задачей.

Характеристика работы обучающегося

Выпускная работа выполнена в полном объеме. Студент освоил методы разработки приложений с использованием параллельных алгоритмов, алгоритмов выделения и обработки графов, анализа литературных источников. Никитин Д.О. обладает высокой степенью ответственности, способностью выполнять работы самостоятельно. В результате выполнения ВКР разработано приложение с параллельным алгоритмом обработки файлов. **Замечания по работе обучающегося**

В выпускной работе подробно описаны настройки среды разработки, блок схема работы, но тестирование программы на корректность и ошибки не были проведены. Выполнен анализ производительности. Это затрудняет оценку качества разработки. В качестве основного подхода использована параллельная обработка входных файлов, а алгоритмов обработки графов последовательные. Также в тексте работы имеются ошибки. Указываются замечания влияют на результаты работы и итоговую оценку.

Допуск к защите

Выпускная квалификационная работа Никитин Д.О. по теме «Разработка и реализация алгоритмов параллельной обработки изображений для суперкомпьютера» отвечает основным требованиям, предъявляемым к квалификационным работам выпускника университета по направлению 09.03.02 «Информационные системы и технологии» и может быть рекомендована к защите.

Оценка труда выпускника

Работу Никитин Д.О. оцениваю на хорошо. При успешной защите ВКР ему может быть присвоена квалификация бакалавра.

Руководитель ВКР: профессор, д.ф.-м.н., доцент

Пустовалов Е.В.

26.06.2022

