МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

ПРОЕКТИРОВАНИЕ СИСТЕМ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ С ИСПОЛЬЗОВАНИЕМ MPI.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ
ПО ДИСЦИПЛИНЕ «Компьютерные системы и сети»

КП.ПО5.170154 - 03 81 00

Листов 12

Руководитель	Савицкий Ю. В.
Выполнил	Корнасевич И. Д.
Консультант по ЕСПД	Савицкий Ю. В.

Брест 2021

Содержание

Введение. Анализ задачи проектирования

С появлением многопроцессорных компьютеров параллельное программирование играет важнейшую роль в обработке информации. Сегодня уже невозможно представить процессор только с одним ядром или сервер работающий в однопоточном режиме. Поэтому понимание многопоточности — это полезное умение любого программиста. Большинство операционных систем, особенно интерактивных, работают сразу на всех ядрах процессора одновременно. Также ОС предоставляет интерфейс взаимодействия с потоками и процессами, а также управление их жизненным циклом. В языке C++, как в одном из наиболее низкоуровневых достаточно инструментов работы с потоками:

а) MPI позволяет работать не только в пределах одной машины, но и связывать много процессов в единый кластер.

б) OpenMP предоставляет наиболее простой интерфейс распараллеливания C++ кода, поэтому я предпочитаю использовать его.

в) Thread родной для C++ модуль, но он слегка многословен, поэтому я решил от него отказаться.

На самом деле неважно какие именно инструменты используются при распараллеливании алгоритмов и программ, принципы от этого не меняются.

Задача разработанной программы — обработка изображений. Задача сводится к применению нескольких алгоритмов к матрице байт. Если алгоритм не сложен, то распараллелить его не составит никакого труда.

1 Разработка и описание последовательного алгоритма программной системы

Исходное изображение представляет собой матрицу пикселей, где каждый элемент имеет значение в пределах **[0, 255]**. С такой матрицей работать очень удобно.

1.1 Общая схема алгоритма.

Схема включает в себя все этапы обработки начиная с загрузки изображения в программу и заканчивая сохранением программы на диске.

а) Загружаем изображение в матрицу байт

б) Определяем, будет ли использован последовательный алгоритм или параллельный

в) Выполняем алгоритм обработки, включающий в себя подавление шумов и улучшение с использованием операций увеличения и уменьшения. Эти алгоритмы используют буферную матрицу пикселей для наиболее корректной обработки изображения.

г) Сохраняем готовое изображение в файловой системе

1.2 Подавление шумов.

Подавление шумов с использованием метода усреднения значений задача не слишком сложная. Так как данные представляют собой матрицу пикселей, задача сводится к простому итерированию по матрице (исключая её края) и изменению значения каждого пикселя в зависимости от значений его соседей. А именно эти значения складываются в переменную аккумулятор, потом значение

аккумулятора делится на 8 (эту операцию можно заменить битовым сдвигом на 3 влево) и это финальное значение помещается в буферную матрицу, так как изменение изначальной матрицы во время выполнения алгоритма снижает его точность. При этом крайние пиксели изображения не обрабатываются сами, но используются для обработки своих соседей.

1.3 Улучшение изображения используя увеличение и уменьшение.

Данная операция не столь однозначна, как подавление шумов. Дело в том, что увеличения у уменьшения можно делать многократно и в разном порядке. Также эти операции используют пороговые значения. Поэтому это уж очень индивидуально для каждого изображения.

В целом, уменьшение представляет собой простое итерирование во всем пикселям картинки, где каждый пиксель становится чёрным (его значение становится равно 0), если он имеет хоть одного соседа, значение которого меньше порогового. Очень важно использовать буферную матрицу при таком преобразовании, результат обработки будет совершенно ужасным, а именно всё изображение рискует стать полностью чёрным.

Увеличение очень похоже на уменьшение, только пиксель становится белым (значение 255), если у него есть сосед, значение которого выше порогового.

2 Разработка программной системы, реализующей последовательный алгоритм обработки

Разработать программу обработки изображений на C++ не простая задача. Во-первых, необходимо придумать способ загрузки изображения в программу так, чтобы с загруженным изображением было удобно работать. Во-вторых, программа должна быть модульной и гибкой для последующих доработок и распараллеливания.

Для загрузки и сохранения изображения и использовал **bitmap**. Это удобный модуль, хранящий изображение в матрице пикселей в формате **RGB**. Так как картинка заведомо чёрно-белая, то все каналы имеют одинаковое значения, поэтому для формирования матрицы используется зелёный канал.

Интересно, что C++ использует файл как единицу трансляции. Поэтому для достижения модульности совершенно не обязательно создавать классы, структуры или функции лишнего порядка. Взаимодействие **.h** и **.cpp** файлов само по себе может обеспечить достаточный уровень инкапсуляции и необходимую гибкость, как если бы я использовал класс через интерфейс. Поэтому, дабы не усложнять код лишний раз, используется необычный подход C++ к файлам.

Как я уже сказал, модульность достигается за счет совместного использования **.h** и **.cpp** файлов.

.h — интерфейс функций загрузки картинки, её обработки и сохранения. Также этот файл содержит размер изображения как константу.

.cpp — реализация интерфейса. Этот файл содержит определение всех функций, также он хранит в себе матрицу пикселей и буферную матрицу пикселей, а также дополнительные функции и определения типов, необходимых для внутреннего использования.

2.1 Тонкости работы функций обработки изображения.

Стоит сказать, что функции обработки изображений используют пару оптимизаций и элементы функционального программирования.

А именно, после обработки основной матрицы с использованием буферной матрицы, данные из буферной матрицы нужно скопировать обратно в основную. Это копирование реализовано при помощи функции **memcpy()**, что намного быстрее, чем поэлементное копирование.

Определение, является ли сосед достаточно ярким или достаточно тёмным в функциях увеличения/уменьшения использует функцию как параметр. Эта функция имеет следующую сигнатуру: **bool func(u_char value)**.

3 Разработка и обоснование варианта схемы параллелизма алгоритма

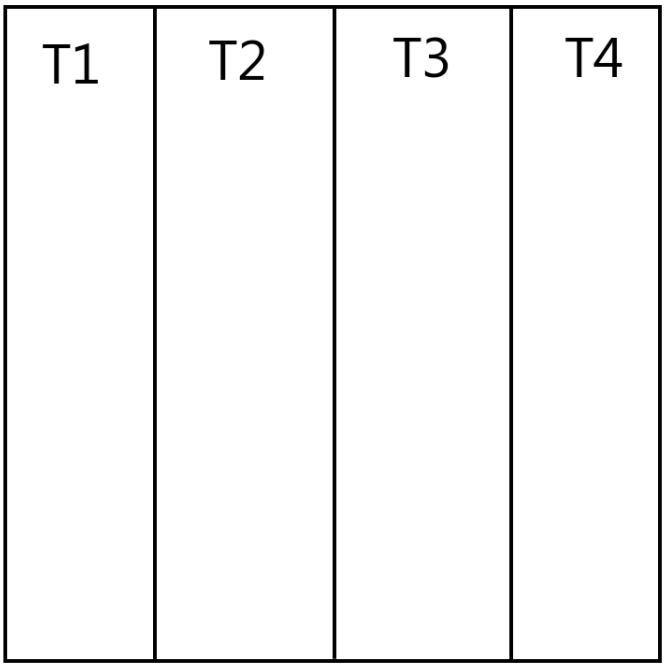

Рисунок 1 – Схема разделения изображения между потоками выполнения.

Как показано на рисунке 1 изображение делится между потоками очень несложно. Всего есть 4 потока: T1, T2, T3, T4. Каждый поток выполнения получает в своё распоряжение четверть исходной картинки. Причём она делится не на квадраты, а на полосы, так как при таком подходе сильно упрощается кодирование и появляется возможность добавить сколь угодно много потоков. При этом не возникает гонок процессов на ресурсы. Каждый процесс работает только в отведенной ему области основной и буферной матрицы. Проблемы могут появиться только на стыке областей ответственности двух процессов, но даже это невозможно, так ка используется буферная матрица. Поэтому изображение будет точно так же, как при использовании однопоточной системы.

Для реализации многопоточности, необходимо изменить функции обработки изображения таким образом, чтобы они могли работать с частью матрицы. Реализуется это очень просто, нужно добавить аргумент начала и конца текущего блока обработки и передать их в корневой `for`. Так как в **bitmap** первый

параметр — x, а второй — y, то разделение получается вертикальное.

Не менее важно вовремя синхронизовать потоки выполнения. Если этого не сделать, то на финальном изображении могут появиться артефакты на стыках работы двух потоков. Поэтому, перед выполнением уменьшения и увеличения потоки должны синхронизоваться.

4 Разработка программных модулей системы параллельной обработки данных

Для реализации алгоритма была выбрана библиотека OpenMP [4], хотя в задании значится MPI [3].Оказалось, что на практике из-за того, что MPI работает с процессами, он не поддерживает shared memory, лишь distributed memory. Это значит, что у каждого процесса своя собственная, только ему принадлежащая память. Из-за этого приходится несколько раз открывать файл с изображением, несколько раз его читать, а для записи передать матрицы размером несколько мегабайт. Я уже не говорю о том, что некоторые алгоритмы (устранение шумов, например) буквально требуют доступа к чужой памяти, иначе на изображении появляются артефакты в виде вертикальных полос. Также резко подскакивает количество чисто технического кода, что неизбежно ведёт к багам.

Оказалось, что реализация на OpenMP совсем не сложна:

а) необходимо добавить `#include <omp.h>` к заголовкам.

б) вызвать в `main()` функцию `omp_set_num_threads(THREAD_COUNT)`, где `THREAD_COUNT` — количество потоков.

в) использовать директиву `#pragma omp parallel for default(none)` для распараллеливания цикла.

г) компилировать с флагом `-pthread` для работы OpenMP.

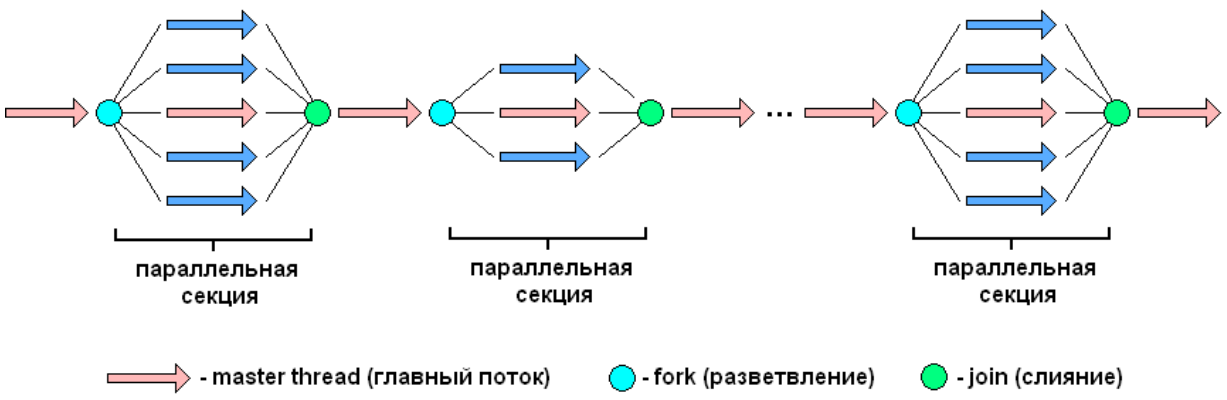

Рисунок 2 – Схема работы OpenMP.

Заключение

Была разработана система улучшения изображений методами удаления шумов, увеличения и уменьшения. Система может работать последовательно и параллельно. Параллельный режим в среднем быстрее в 4 раза.

Были изучены основные принципы параллельного программирования с использованием MPI и OpenMP. Изучены способы работы с изображениями формата bmp в C++ и система сборки CMake.

Список литературы

1. C++ Reference [электронный ресурс]. — URL: http://en.cppreference.com.
2. GitHub [электронный ресурс]. — URL: https://github.com.
3. MPI справочник [электронный ресурс]. — URL: https://www.mpi-forum.org/docs.
4. OpenMP справочник [электронный ресурс]. — URL: https://www.openmp.org/resources/refguides.
5. Stack Overflow [электронный ресурс]. — URL: https://stackoverflow.com.
6. ГОСТ 19.701-90 ЕСПД [электронный ресурс] : Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. — URL: https://znaytovar.ru/gost/2/GOST_1970190_ESPD_Sxemy_algori.html.
7. ГОСТ 7.1-2003 [электронный ресурс] : Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. — URL: https://www.internet-law.ru/gosts/gost/1560.