Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н. Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к курсовой работе на тему:

Реализации инструмента анализа параллельных программ

Студент Капустин А.И. И.О. Фамилия
(Подпись, дата)

Руководитель курсового проекта Ковтушенко А.П. И.О. Фамилия
(Подпись, дата)

Москва 2017

Содержание

Введение

Вычислительные системы с массовым параллелизмом совершили революцию в мире параллельных вычислений, обеспечив производительность порядка нескольких Терафлоп/с. Необходимость в вычислениях такого объёма возникает в широком круге областей от молекулярной физики до предсказания погоды. На сегодняшний день массивно параллельные системы являются лидерами по критерию цена/производительность. Однако, за таким прогрессом аппаратного обеспечения не поспевало программное обеспечение. Очевидно, что будут созданы более мощные массивно параллельные системы, и основной проблемой становится разработка программного обеспечения, способного максимально использовать вычислительные ресурсы. Параллельные вычисления связаны с очень сложным и тяжело понимаемым ходом выполнения программ, и недостаточно полное понимание не позволяет полностью использовать вычислительные ресурсы. Так как производительность – самый важный критерий при использовании параллельных компьютеров, и так как производительность пилотных реализаций параллельных программ как правило далека от ожидаемой, модификация программы с целью повышения производительности является важной частью процесса разработки. Основным фактором, ведущим к большому количеству трудоёмкой работы, требующей высокой квалификации является нехватка средств для анализа и оценки производительности. При отсутствии таких средств, причины низкой производительности выявляются с помощью средств, разработанных для конкретного приложения и метода тыка. Повышение производительности параллельных программ наиболее важная, но трудоёмкая задача, которая требует хорошего понимания задачи. Цель средств контроля и визуализации производительности параллельных программ – помочь разработчику обрести это понимание. Визуализация производительности заключается в использовании графических образов для представления анализа данных о выполнении программы для лучшего её понимания. Такие средства визуализации были очень полезны и в прошлом, и широко используются в настоящее время для повышения производительности параллельных программ. Несмотря на прогресс в области визуализации научных данных, технологии визуализации для параллельного программирования являются объектом внимания многих разработчиков, так как требуется тем более сложная визуализация, чем более сложными становятся параллельные системы.

1 Аналитический раздел

Алгоритмы, с которыми приходится иметь дело при параллельном программировании гораздо сложнее, чем аналогичные последовательные алгоритмы. Часть проблем касается распараллеливания самого алгоритма (например, как разбить процесс на задачи и как поставить им в соответствие физические процессоры), часть касается оптимального использования вычислительных ресурсов (баланс загрузки процессоров, межпроцессные взаимодействия, доступ к кэшу). Для разработки высокопроизводительных приложений программисту не только очень хорошо знать структуру самого приложения, но и структуру аппаратной платформы, а зачастую и программной (особенности операционной системы, компилятора и.т.д.). Итак, уровень знаний, необходимый для разработки высокопроизводительных параллельных программ выше, чем для последовательных программ. Параллельное программирование может быть очень мучительным и запутанным. В добавок отладка параллельной программы и поиск критических мест также трудоёмкий и длительный процесс. Из-за этих причин программирование для массивно параллельных машин считается непростым делом, и в значительной степени причиной этому служит недостаток инструментария по сравнению с последовательными машинами. Для того, чтобы понять поведение и внутреннюю логику параллельных программ и помочь программисту выявить потенциальные слабые места (например, ограничения по межпроцессным взаимодействиям) необходимы средства контроля хода выполнения параллельной программы. Они могут использоваться для сравнения производительности схожих программ или помогать при отладке.

1.1 Парадигмы параллельного программирования

1.1.1 Структура и характеристики параллельных алгоритмов

Любой параллельный алгоритм (программа) состоит из блоков последовательных и параллельных вычислений. Последовательная часть программы (называется критическим сечением) – это последовательность операторов, которая должна выполняться только одним процессором. За критическим сечением обычно следует ветвление, инициирующее параллельно выполняемые участки программы (параллельные сегменты). В месте соединения параллельных сегментов выполняется синхронизация и параллельные сегменты возвращаются к критическому сечению. Синхронизация требуется для того, чтобы вычисления в параллельных сегментах закончились прежде, чем начнется выполнение последовательной части.

1.1.2 Программные парадигмы

При параллельном программировании появляются дополнительные сложности (по сравнению с последовательным программированием): возникает необходи-

мость управлять несколькими процессорами и координировать межпроцессорные вызовы. В общем случае параллельная программа представляет собой ряд заданий, работающих параллельно друг другу и взаимодействующих между собой. Существует несколько парадигм (моделей) программирования при создании параллельных алгоритмов. Наиболее распространенными парадигмами являются модель передачи сообщений и модель разделяемой памяти.

1.1.3 Проектирование параллельных алгоритмов

Несмотря на то, что проектирование параллельных алгоритмов есть процесс творческий и в общем случае достаточно сложно рекомендовать универсальный рецепт, следует все же привести общую методологию проектирования параллельных алгоритмов, которая в значительной степени оградит проектировщика от потенциальных ошибок.

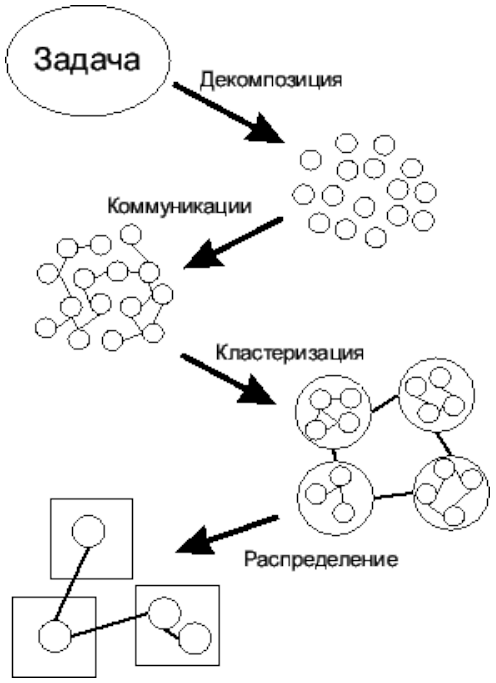

Рисунок 1.1 — Методология проектирования параллельных алгоритмов

Описываемая методология проектирования предлагает подход к распараллеливанию, который рассматривает машинно-независимые аспекты реализации алгоритма, такие как параллелизм, на первой стадии, а особенности проектирования, связанные с конкретным параллельным компьютером, - на второй. Данный подход разделяет процесс проектирования на четыре отдельных этапа: декомпозиция (partitioning), коммуникации (communications), кластеризация (agglomeration) и распределение (mapping). Первые два этапа призваны выделить в исходной задаче параллелизм и масштабируемость, остальные этапы связаны с различными аспектами

производительности алгоритма. Рисунок 1.1 иллюстрирует процесс проектирования параллельного алгоритма. Сформулируем кратко содержание этих этапов.

1.1.3.1 Декомпозиция

Общая задача вычислений и обработки данных делится на меньшего размера подзадачи. При этом игнорируются проблемы практической реализации, например, число процессоров используемого в будущем компьютера. Напротив, все внимание сосредотачивается на возможном параллелизме исходной задачи.

1.1.3.2 Коммуникации

Определяется требуемая связь между подзадачами, ее структура и конкретные алгоритмы коммуникаций.

1.1.3.3 Кластеризация

Структура подзадач и коммуникаций оценивается с учетом требований производительности алгоритма и затрат на реализацию. Если необходимо, отдельные подзадачи комбинируются в более крупные с целью улучшения производительности и снижения затрат на разработку.

1.1.3.4 Распределение

Каждая подзадача назначается определенному процессору, при этом стараются максимально равномерно загрузить процессоры и минимизировать коммуникации. Распределение может быть статическим или формироваться в процессе выполнения программы на основе алгоритмов балансировки загрузки (load balancing algorithms). Следует заметить, что при прохождении последних двух этапов необходимо учитывать архитектуру параллельного компьютера, имеющегося в распоряжении разработчика, другими словами параллельный алгоритм должен быть адекватен используемой параллельной системе.

1.2 Дизеринг при помощи диффузии ошибок

Метод, обладающий наилучшим качеством среди представленных, - метод рассеивания ошибок. Но так же он, к сожалению, самый медленный.[?] Существуют несколько вариантов этого алгоритма, причем скорость алгоритма обратно пропорционально качеству изображения.[?] Суть алгоритма: для каждой точки изображения находим ближайший возможный цвет. Затем мы рассчитываем разницу между текущим значеним и ближайшим возможным. Эта разница и будем нашем значением ошибки.Это значение ошибки мы распределяем между соседними элементами, кото-

рые мы ещё не посещали. Для последних точек ошибка распределяется между уже посещенными точками.

1.2.1 Характеристики производительности параллельного алгоритма

Задачей программиста является проектирование и реализация программ, которые удовлетворяют требованиям пользователя в смысле производительности и корректной работы. Производительность является достаточно сложным и многосторонним понятием. Кроме времени выполнения программы и масштабируемости следует также анализировать механизмы, отвечающие за генерацию, хранение и передачу данных по сети, оценивать затраты, связанные с проектированием, реализацией, эксплуатацией и сопровождением программного обеспечения. Существуют весьма разнообразные параметры, влияющие на производительность вычислительной системы. Среди них наиболее значимыми являются: требования по памяти, производительность сети, время задержки при передаче данных (latency time), переносимость, масштабируемость, затраты на проектирование, реализацию, отладку, требование к аппаратному обеспечению и т.д. Критериями, по которым оценивается производительность, являются время выполнения программы, ускорение и эффективность. Относительная важность различных параметров будет меняться в зависимости от природы вычислительной системы и поставленной задачи. Специфика конкретной задачи может требовать высоких показателей для каких-либо определенных критериев, в то время как остальные должны быть оптимизированы или полностью игнорированы. Ниже мы рассмотрим лишь две характеристики производительности, а именно, эффективность и ускорение, поскольку они наиболее часто используются при анализе производительности параллельных алгоритмов.

$$sss$$

(1.1)

1.2.2 Фильтр Флойда-Стейнберга

Каждый пиксель распределяет свою ошибку на соседние с ним пиксели. Коэффициенты были подобраны таким образом, что в районах с интенсивностью 1/2 от общего количество оттенков, изображение выглядело похожим на шахматную доску.

$$\begin{vmatrix} \boxminus & \boxtimes & 7 \\ 3 & 5 & 1 \end{vmatrix} \quad (1/16)$$

1.2.3 "Ложный"фильтр Флойда-Стейнберга

В случае сканирования слева-направо этот фильтр порождает большое количество артефактов.Чтобы получить изображение с меньшим количеством артефактов, нужно чётные строки сканировать справа-налево, а нечетные строки сканировать слева-направо.

$$\begin{vmatrix} \boxtimes & 3 \\ 3 & 2 \end{vmatrix} (1/8)$$

1.2.4 Фильтр Джарвиса,Джунка и Нинка

В случае когда фильтры Флойда-Стейдберга дают недостаточно хороший результат, применяются фильтры с более широким распределением ошибки. Фильтр Джарвиса, Джунка и Нинка требует связи с 12 соседями, что очевидно ведет в большим затратам памяти и времени[?]:

$$\begin{vmatrix} \boxminus & \boxminus & \boxtimes & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{vmatrix} (1/48)$$

1.2.5 Фильтр Стаки

Фильтр разработан на основе фильтра Джарвиса, Джунка и Нинка.После такого как мы вычислим 8/42 ошибки, остальные значения можно получить при помощи побитовых сдвигов, тем самым сокращая время работы алгоритма.

$$\begin{vmatrix} \boxminus & \boxminus & \boxtimes & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{vmatrix} (1/42)$$

1.2.6 Фильтр Бурка

Стаки. Результат можно получить чуть быстрее за счет использования побитовых операций.

$$\begin{vmatrix} \boxminus & \boxminus & \boxtimes & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \end{vmatrix} (1/32)$$

Существует много различных вариантов фильтро дизеринга при помощи диффузии ошибок, здесь приведены наиболее популярные алгоритмы.[?]

1.3 Прочие виды дизеринга

Компьютерная графика не стоит на месте и завидной переодичностью появляются всё новые виды дизеринга, не укладывающиеся в привычную классификацию.

1.3.1 Алгоритм Юлиомы

Существует три вариации этого алгоритма [?] значительно отличающиеся друг от друга. Основная идея этих алгоритмов заключается в оптимальном подборе цвета ограниченной цветовой палитры, основываваясь на визуальном восприятии этого цвета, а не на его числовом коде. Отличается от остальных алгоритмов значительными временными затратами, что делает применение этого алгоритма для решения практических задач затруднительным.

1.4 Выбор оптимального класса алгоритма

Вышеприведенные методы упорядочены по качеству получаемого на выходе изображения, однако, такие соображения как время, экономия памяти и прочие являются определяющими при выборе алгоритма[?]. На основе вышеприведенных данных сравним классы алгоритмов дизеринга.

Характеристика / Вид алгоритма	Скорость	Качество	Доп память
Случайный	+	-	-
Шаблонный	+-	-+	+
Упорядоченный	-+	+-	-
Диффузия ошибок	-	+	+
Остальные	-	+	+

Самым быстрым классом алгоритмов дизеринга является случайный дизеринг.Однако, качество получаемых при помощи изображений низко. Алгоритмы упорядоченного дизеринга позволяют получить изображения более высокого качества, однако работают более медленно и требуют дополнительных затрат. В зависимости от вычислительных ресурсов и требуемого результата рациональным будет применение различых алгоритмов.

1.5 Оценка алгоритмов

n - количество пикселей в изображении

Характеристика / Вид алгоритма	Время	Память
Случайный	O(n)	O(1)
Упорядоченный	O(n)	O(1)
Диффузия ошибок	O(n)	O(n)
Первый алгоритм Юлиомы	O(n^3)	O(n)

2 Конструкторский раздел

2.1 Оценка качества изображений

2.1.1 PSNR

Пиковое отношение сигнала к шуму (англ. peak signal-to-noise ratio) - соотношение между максимумом возможного значения сигнала и мощностью шума, искажающего значения сигнала.[?]

$$PSNR = 20\log_{10}\frac{MAX_i}{\sqrt{MSE}} \tag{2.1}$$

Где MAX_i - это максимальное значение, принимаемое пикселем изображения, MSE - среднеквадратичное отклонение. Для двух монохромных изображений I и K размера m×n, одно из которых считается зашумленным приближением другого, вычисляется так:

$$MSE = \frac{1}{m*n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}|I(i,j) - K(i,j)|^2 \tag{2.2}$$

2.1.2 SSIM

Индекс структурного сходства (SSIM от англ. structure similarity) — метод измерения схожести между двумя изображениями путем полного сопоставления. SSIM-индекс является развитием традиционных методов, таких как PSNR (peak signal-to-noise ratio) и метод среднеквадратичной ошибки MSE, которые оказались несовместимы с физиологией человеческого восприятия.

Отличительной особенностью метода, в отличие от MSE и PSNR, является то, что он учитывает «восприятие ошибки» благодаря учёту структурного изменения информации. Идея заключается в том, что пиксели имеют сильную взаимосвязь, особенно когда они близки пространственно. Данные зависимости несут важную информацию о структуре объектов и о сцене в целом. Особенностью является, что SSIM всегда лежит в промежутке от -1 до 1, причем при его значении равном 1, означает, что мы имеем две одинаковые картинки. Общая формула имеет вид

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(\sigma_x y + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.3}$$

Тут μ_x среднее значение для первой картинки, μ_y для второй, σ_x среднеквадратичное отклонение для первой картинки, и соотвественно σ_y для второй, $\sigma_x y$ это уже ковариация. Она находится следующим образом:

$$\sigma_x y = \mu_x y - \mu_x\mu_y \tag{2.4}$$

c_1 и c_2 - поправочные коэффициенты, которые нужны вследствие малости знаменателя.

$$c_1 = (0{,}01 * d)^2 \tag{2.5}$$

$$c_2 = (0{,}03 * d)^2 \tag{2.6}$$

d - количество цветов, соответствующих данной битности изображения Для подтверждения или опровержения вышеописанных гипотезы реализуются несколько алгоритмов случайного распределения, алгоритм Флойда-Стейнберга, модификации на основе алгоритма Флойда-Стейнберга. Результаты работы алгоритмов сравниваются по времени работы, по количеству затрачиваемой памяти,а так же по SSIM и PSNR.

2.1.3 Алгоритм случайного распределения

P(x,y) - цвет конкретного пикселя

Листинг 2.1 — Алгоритм случайного распределения

```
for x in range(height):
    for y in range(weight):
        if P(x,y)>127:
            P(x,y) = 255
        else:
            P(x,y) = 0
```

2.2 Виды случайных распределений

2.2.1 Белый шум

Белый шумом называют сигнал с равномерной спектральной плотностью на всех частотах и дисперсией, равной бесконечности. Является стационарным случайным процессом. В качестве сигнала в задаче дизеринга рассматриватся последовать последовательность чисел, получаемых от генератора случайных чисел.

2.2.2 Коричневый шум

Спектральная плотность коричневого шума пропорциональна $1/f^2$, где f — частота. Это означает, что на низких частотах шум имеет больше энергии, чем на высоких. То есть пикселей темных цветов большей, чем пикселей светлого цвета.Применение фильтра коричневого шума в целом затемняет получаемое изображение.

Листинг 2.2 — Получение коричневого шума

```
def smoother(noise):
    output = []
    for i in range(len(noise) - 1):
        output.append(0.5 * (noise[i] + noise[i+1]))
return output
```

2.2.3 Гауссовский шум

Гауссовский шум - шум, имеющий функцию плотности вероятности (PDF), равную нормальному распределению, которое также известно как гауссово распределение.

$$p_g(z) = \frac{1}{\sigma\sqrt{2 * \pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{2.7}$$

z - количество цветов, μ -среднее значение, σ - стандартное отклонение.

2.2.4 Фиолетовый шум

Фиолетовый шум представляет собой противооложноть между коричневому шуму. Получение его аналогично получению коричневого шума.Применение фильтра фиолетового шума в целом засветляет получаемое изображение.

Листинг 2.3 — Получение розового шума

```
def rougher(noise):
    output = []
    for i in range(len(noise) - 1):
    output.append(0.5 * (noise[i] - noise[i+1]))
return output
```

2.2.5 Розовый и синий шумы

Розовый и синий шумы представляют собой "промежуточные"шумы.Изобржаение с розовым шумом темнее изображения с белым шумом, но

светлее изображения с коричневым шумом. Изображение с синим шумом светлее изображения с белым шумом, но темнее чем изображение с фиолетовым шумом. Их получение аналогично получению со коричневого и фиолетового шумов.

2.3 Алгоритм Байера

Листинг 2.4 — Алгоритм Флойда-Стейнберга

```
For each pixel, Input,:
    Factor  = ThresholdMatrix[xcoordinate % X][ycoordinate % Y];
    Attempt = Input + Factor * Threshold
    Color = FindClosestColorFrom(Palette, Attempt)
```

2.4 Алгоритм Флойда-Стейнберга

Рассмотрим более детально алгоритм Флойда-Стейнберга. P(x,y) - цвет пикселя в точке x,y I(x,y) - предполгаемый цвет пикселя с учетом ошибки(действительное число)

Следует отметить, что сумма "долей"рассевания ошибки и для других алгоритмов упорядоченного дизеринга будет равна единице.

Листинг 2.5 — Алгоритм Флойда-Стейнберга

```
for x in range(width):
    for y in range(height):
        P(x,y) = trunc(I(x,y)+0.5)
        e = I(x,y) - P(x,y)
        I(x,y-1) += 7/16*e
        I(x+1, y-1) += 3/16*e
        I(x+1, y) += 5/16*e
        I(x+1, y+1) += 1/16*e
```

2.5 Ложный алгоритм Флойда-Стейнберга

Отличительной особенностью ложного алгоритма Флойда-Стейнберга является по пикселям изображения справа-налево, а не слева-направо как в большинстве других алгоритмов.

Листинг 2.6 — Ложный алгоритм Флойда-Стейнберга

```
for x in range(width):
```

```
2       for y in range(height):
3           P(x,y) = trunc(I(x,y)+0.5)
4           e = I(x,y) - P(x,y)
5           P(x,y) = trunc(I(x,y)+0.5)
6           e = I(x,y) - P(x,y)
7           I(x,y-1) += 3/8*e
8           I(x-1, y-1) += 2/8*e
9           I(x-1, y) += 3/8*e
```

2.6 Алгоритм Джарвиса,Джунка и Нинка

Листинг 2.7 — Алгоритм Джарвиса,Джунка и Нинка

```
1  for x in range(width):
2      for y in range(height):
3          P(x,y) = trunc(I(x,y)+0.5)
4          e = I(x,y) - P(x,y)
5          P(x,y) = trunc(I(x,y)+0.5)
6          e = I(x,y) - P(x,y)
7          I(x,y-1) += 3/48*e
8          I(x+1,y+1) += 5/48*e
9          I(x,y+1) += 7/48*e
10         I(x-1,y+1) += 5/48*e
11         I(x-2,y+1) += 3/48*e
12         I(x-1,y+1) += 5/48*e
13         I(x+2,y+2) += 1/48*e
14         I(x+1,y+2) += 3/48*e
15         I(x, y+2) += 5/48*e
16         I(x-1,y+2) += 3/48*e
17         I(x-2,y+2) += 1/48*e
18         I(x+1,y) += 7/48*e
19         I(x+2,y) += 5/48*e
```

2.7 Первый алгоритм Юлиомы

Листинг 2.8 — Первый алгоритм Юлиомы

```
1  For each pixel, Input, in the original picture:
2      Factor  = ThresholdMatrix[xcoordinate % X][ycoordinate % Y];
3      Make a Plan, based on Input and the Palette.
4
5      If Factor < Plan.ratio,
```

```
6           Draw pixel using Plan.color2
7       else,
8           Draw pixel using Plan.color1
```

Листинг 2.9 — План нахождения цвета пикселя

```
1  SmallestPenalty = 10^99
2  For each unique combination of two colors from the palette, Color1
       and Color2:
3       For each possible Ratio, 0 .. (X*Y-1):
4           Mixed = Color1 + Ratio * (Color2 - Color1) / (X*Y)
5           Penalty = Evaluate the difference of Input and Mixed.
6
7           If Penalty < SmallestPenalty,
8           SmallestPenalty = Penalty
9           Plan = { Color1, Color2, Ratio / (X*Y) }1
```

Листинг 2.10 — Вспомогательная матрица перого Алгоритма Юлиомы

```
1       [0/64, 48/64, 12/64, 60/64, 3/64, 51/64, 15/64, 63/64,
2       32/64, 16/64, 44/64, 28/64, 35/64, 19/64, 47/64, 31/64,
3       8/64, 56/64, 4/64, 52/64, 11/64, 59/64, 7/64, 55/64,
4       40/64, 24/64, 36/64, 20/64, 43/64, 27/64, 39/64, 23/64,
5       2/64, 50/64, 14/64, 62/64, 1/64, 49/64, 13/64, 61/64,
6       34/64, 18/64, 46/64, 30/64, 33/64, 17/64, 54/64, 29/64,
7       10/64, 58/64, 6/64, 54/64, 9/64, 57/64, 5/64, 53/64,
8       42/64, 26/64, 38/64, 22/64, 41/64, 25/64, 37/64, 21/64]
```

3 Технологический раздел

3.1 Выбор языка программирования

Для реализации данных алгоритмов был выбран язык C++. Данный язык был обоснован следующими причинами: Причины:

а) Компилируемый язык со статической типизацией.

б) Сочетание высокоуровневых и низкоуровневых средств.

в) Реализация ООП.

г) Наличие удобной стандартной библиотеки шаблонов

3.2 Выбор вспомогательных библиотек

Для реализации программы была выбрана библиотека Qt.

а) Широкие возможности работы с изображениями, в том числе и попиксельно

б) Наличии более функциональных аналогов стандартной библиотеки шаблонов в том числе для разнообразных структур данных

Так же были были использованы библиотеки ImageMagick(для конвертации изображения в ограниченную цветовую палитру), OpenMP(многопоточность), OpenGL(работа с шейдерами)

3.2.1 Диаграмма классов

Для реализации различных алгоритмов была разработана следующая структура классов. Был создан абстракный класс дизеринга Dithering с интерфейсом, наследуюмом в дочерних классах. Так же была введена система менджеров: DitherManager, MetricsManager, DataManager и MainManager.

4 Исследовательский раздел

4.1 Время дизеринга раличных алгоритмов

Рассмотрим время работы различных алгоритмов для различных размеров изображения.

	Размер, пиксели	Время, мкс
White noise	133x90	862
Blue noise	133x90	930
Brown noise	133x90	934
Violet noise	133x90	937
Pink noise noise	133x90	930
Floyd-SD	133x90	1200
F. Floyd-SDe	133x90	1093
JJN	133x90	1909
White noise	458x458	15735
Blue noise	458x458	19374
Brown noise	458x458	19432
Violet noise	458x458	18787
Pink noise noise	458x458	18129
Floyd-SD	458x458	27173
F. Floyd-SDe	458x458	26424
JJN	458x458	47201
White noise	458x458	194376
Blue noise	458x458	200577
Brown noise	458x458	208400
Violet noise	458x458	251294
Pink noise noise	458x458	258775
Floyd-SD	458x458	251294
F. Floyd-SDe	458x458	387104
JJN	458x458	857481

Из рассмотрения вынесены алгоритм Юлиомы в вследствие того, что он значительно медленней других алгоритмов(2732568 мкс для изображения 113x90) в и алгоритм Байера, реализованный при помощи шейдеров, вследствии того, что он не не укладывается в рамки требуемой палитры (при этом он работает очень быстро 64 мс для изображени 640x480).

4.2 Качество получаемого изображения

	PSNR	SSIM
White noise	33.2894	0.914778
Blue noise	36.1756	0.971626
Brown noise	33.32370	0.915767
Violet noise	37.63480	0.984574
Pink noise	36.4484	0.974718
Floyd-SD	37.0553	0.979173
F. Floyd-SDe	36.8401	0.976452
JJN	37.30740	0.981688
Yliouma	36.2359	0.967796
Without dithering	37.6348	0.984574

Несмотря на то, что некоторые сложные алгоритмы дизеринга диффузии ошибок обещат получения хорошего качества изображений,некоторые алгоритмы случайного дизеринга на конкретных изображениях дают лучший результат. Для того чтобы получить наилучший результат дизеринга, следует проанализировать результаты дизеринга нескольких изображений и выбрать среди них наилучшее. Так же следует отметить некоторую необъективность метрик: результат метрик не всегда совпадает с человеческим восприятием картинки.

4.3 Размер получаемого изображения

	Разрешение, пикс	Размер, кб	Исх. раз., кб
White noise	900x675	186	2373 bmp,1779 png
Blue noise	900x675	135	
Brown noise	900x6750	186	
Violet noise	900x675	98	
Pink noise	900x675	1158	
Floyd-SD	900x675	1273	
F. Floyd-SDe	900x675	143	
JJN	900x675	117	
White noise	3984x32355	3431	50344 bmp,37758 png
Blue noise	3984x3235	2570	
Brown noise	3984x3235	3432	
Violet noise	3984x3235	1950	
Pink noise	3984x3235	2406	
Floyd-SD	3984x32355	3605	
F. Floyd-SDe	3984x3235	4269	
JJN	3984x3235	3716	

Из вышеприведенной таблицы, можно заметить, размер изображения после дизеринга значительно уменьшиается,достирается выигрыш в размере изображение до 15 раз, в зависимости от иходного контейнера изображения и выбранного способра дизеринга.

Заключение

В данной работе были реализованы различные алгоритмы дизеринга, было произведено сравнение и анализ этих алгоритмов. Программа позволяет получить изображение схожего визуального качества при значительном уменьшении размера. Был получен вывод о том, что для различных целей следует использовать различные алгоритмы дизеринга, универсального алгоритма дизеринга не существует. Программа не привязана к какой-то конкретной операционной системе и может быть скомпилирована и запущена на всех популярных ОС.