Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра систем искусственного интеллекта

УТВЕРЖДАЮ
Заведующий кафедрой СИИ

_____ Г. М. Цибульский
«21» 06 2016г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Двумерные дискретные преобразования Фурье с узлами на центрированной решетке

09.04.02. Информационные системы и технологии

09.04.02.07 Информационно-управляющие системы

Руководитель проф., д-р.физ.-мат.наук М. В. Носков

Студент КИ14-05-1М 031404383 И. С. Мерко

Рецензент проф., д-р.техн.наук К. В. Симонов

Нормоконтролер 01.06.16 доцент М. А. Аникьева

Красноярск 2016

СОДЕРЖАНИЕ

Введение …… 4
1 Постановка задачи ……. 6
1.1 Современное состояние проблемы ……... 6
1.2 Актуальность задачи …….. 10
2 Вводная глава ……. 12
2.1 Точная постановка задачи ……. 12
2.2 Преобразование Фурье…… 12
2.3 Двумерное дискретное преобразование Фурье и его обращение …….. 15
2.4 Вычисление двумерного дискретного преобразования Фурье …….. 18
2.5 Вычисление трёхмерного дискретного преобразования Фурье ……. 25
2.6 Описание библиотеки MPI …… . . 28
2.7 Комплекс высокопроизводительных вычислений ИКИТ СФУ ……. 31
3 Основные достигнутые результаты ……. 33
3.1 Словесное описание алгоритма работы программы …….. 33
3.1 Блок-схема алгоритма работы программы ……... 35
3.2 Основные трудности, возникшие в процессе работы ……. 36
3.3 Результаты численного эксперимента в двумерном случае …….. 41
3.4 Результаты численного эксперимента в трёхмерном случае …… ... 42
Заключение …… 43
Список использованных источников …….. 44
Приложение А. Текст программы двумерного БПФ …… 46
Приложение Б. Текст программы трёхмерного БПФ …… 67

ВВЕДЕНИЕ

Одним из сопутствующих факторов компьютерной революции оказалось появление совершенно новых областей исследования. С каждым годом по мере увеличения быстродействия, уменьшения стоимости и размеров ИС растут возможности решения задач все возрастающей сложности. К ним относится цифровая обработка многомерных сигналов, требующая значительных объемов цифровой памяти и соответствующего количества арифметических операций. Несмотря на сложность, цифровая обработка сигналов уже позволила найти решение ряда важных задач, начиная с компьютерной томографии (методики, позволяющей по проекциям рентгеновского изображения, полученным при различных ориентациях детекторов, выполнять трехмерную реконструкцию органов человеческого тела) и кончая проектированием полей пассивных акустических датчиков и исследованием ресурсов Земли с помощью спутников. Цифровая обработка многомерных сигналов имеет также прочное математическое обоснование, позволяющее не только понять уже достигнутое, но и эффективно исследовать новые проблемы по мере их возникновения и успешно их решать.

Сигнал - это некоторое средство для передачи информации, а целью обработки сигналов является извлечение этой информации. Так, ансамбли изменяющихся во времени электрических потенциалов, плотность зерен серебра фотографической эмульсии или массивы чисел в памяти ЭВМ представляют собой примеры сигналов. Обычно обработка сигналов включает в себя перенос информации с одного сигнала на другой. Независимо от своей физической сущности сигналы представляют интерес только благодаря содержащейся в них информации. Можно сказать, что обработка сигналов включает в себя две основные задачи - преобразование способа представления информации в сигнале и сокращение ее объема.

Цифровая обработка сигналов касается обработки сигналов, которые можно представить в виде последовательности чисел, а цифровая обработка

многомерных сигналов - обработки сигналов, представленных в виде многомерных массивов чисел, например массивов, получаемых после дискретизации изображений или результатов дискретизации непрерывно изменяющихся во времени сигналов, поступающих одновременно от нескольких датчиков.

1 Постановка задачи

1.1 Современное состояние проблемы

На сегодняшний день в вычислении быстрого преобразования Фурье (БПФ) наблюдаются следующие тенденции:

Аппаратное ускорение: использование спецпроцессоров и вычислительной мощности видеокарт.

Рост частот универсальных процессоров натолкнулся на физические ограничения и высокое энергопотребление, и увеличение их производительности всё чаще происходит за счёт размещения нескольких ядер в одном чипе. Продаваемые сейчас процессоры содержат лишь до четырёх ядер (дальнейший рост не будет быстрым) и они предназначены для обычных приложений, используют MIMD - множественный поток команд и данных. Каждое ядро работает отдельно от остальных, исполняя разные инструкции для разных процессов.

Специализированные векторные возможности (SSE2 и SSE3) для четырехкомпонентных (одинарная точность вычислений с плавающей точкой) и двухкомпонентных (двойная точность) векторов появились в универсальных процессорах из-за возросших требований графических приложений, в первую очередь. Именно поэтому для определённых задач применение GPU выгоднее, ведь они изначально сделаны для них.

Например, в видеочипах NVIDIA основной блок — это мультипроцессор с восемью-десятью ядрами и сотнями ALU в целом, несколькими тысячами регистров и небольшим количеством разделяемой общей памяти. Кроме того, видеокарта содержит быструю глобальную память с доступом к ней всех мультипроцессоров, локальную память в каждом мультипроцессоре, а также специальную память для констант.

Самое главное — эти несколько ядер мультипроцессора в GPU являются SIMD (одиночный поток команд, множество потоков данных) ядрами. И эти ядра исполняют одни и те же инструкции одновременно, такой стиль программирования является обычным для графических алгоритмов и многих научных задач, но требует специфического программирования. Зато такой подход позволяет увеличить количество исполнительных блоков за счёт их упрощения.

Итак, перечислим основные различия между архитектурами CPU и GPU. Ядра CPU созданы для исполнения одного потока последовательных инструкций с максимальной производительностью, а GPU проектируются для быстрого исполнения большого числа параллельно выполняемых потоков инструкций. Универсальные процессоры оптимизированы для достижения высокой производительности единственного потока команд, обрабатывающего и целые числа и числа с плавающей точкой. При этом доступ к памяти случайный [16].

Ещё один вариант ускорения преобразования Фурье — это использование специализированных процессоров, например, HPCx система, применяемая в университете Эдинбурга (the University of Edinburgh), Великобритания. Она состоит из 160 вычислительных и 8 управляющих файловых серверов IBM eServer с 575 узлами. Каждый узел состоит из 16 1.5 Гц POWER5 64-bit RISC процессоров. Вычислительная мощность системы HPCx составляет 15.3 терафлопс.

Процессор POWER5 64-bit RISC это двухядерный процессор с поддержкой одновременной многопоточности с двумя потоками, то есть фактически он реализует 4 логических процессора. Благодаря виртуальной векторной архитектуре, процессор POWER5 может работать как один векторный процессор.

В университете Эдинбурга ведется работа по распараллеливанию одно-, двух-, и трехмерного преобразования Фурье (многомерное преобразование

Фурье ведется по строчкам и столбцам) при помощи OpenMP и MPI. Результаты экспериментов представлены в таблице [18].

Таблица 1 – Использование OpenMP, Университет Эдинбурга

Время в секундах

Число процессов	64×64	128×128	256×256	512×12	1024×1024	2048×2048	4096×4096	8192×8192
1	0,000119	0,000648	0,003352	0,017743	0,114502	0,599953	3,238710	14,58787
2	0,000073	0,000355	0,001733	0,009318	0,054626	0,328531	1,632488	7,432540
4	0,000047	0,000195	0,000876	0,004702	0,026887	0,184591	0,833760	3,834367
8	0,000038	0,000115	0,000483	0,002515	0,014010	0,101344	0,443282	1,980915
16	0,000029	0,000096	0,000282	0,001589	0,008271	0,057241	0,270873	1,331173

Таблица 2 – Использование MPI, Университет Эдинбурга

Время в секундах

Размер сигнала	MPI
64×64	0,000034
128×128	0,000092
256×256	0,000406
512×512	0,001570
1024×1024	0,009215
2048×2048	0,042280
4096×4096	0,273670
8192×8192	1,235662

Переход от основания 2 к более высоким (4,8,16, …) при использовании одномерного алгоритма БПФ Кули-Тьюки. Например, схема расчёта БПФ по основанию 16 для сигнала с 256 отсчётами. При использовании основания 2 потребуется 8 рекурсивных шагов, при использовании основания 16 – всего 2 (рисунок 1, 2).

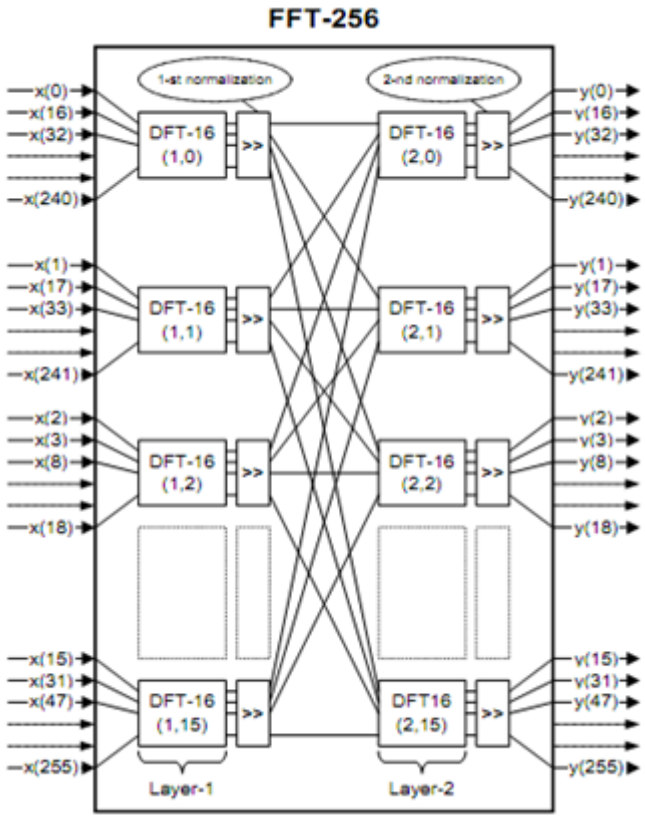

Рисунок 1 – Обобщенный граф вычислений 256-точечного быстрого преобразования Фурье (БПФ-256/FFT-256) по основанию 16

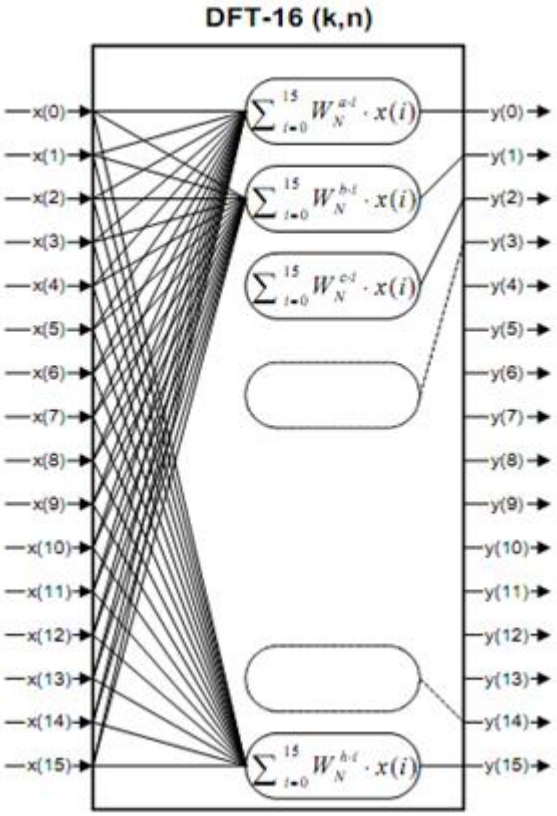

Рисунок 2 – Развернутая схема блока 16-точечного дискретного преобразования Фурье (ДПФ-16/DFT-16)

Разбиение по строчкам и столбцам и вычисление их одномерного преобразования Фурье для вычисления двумерного.

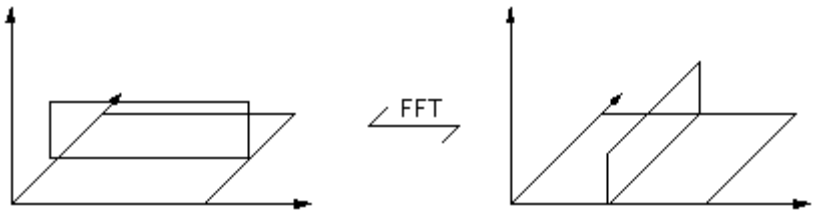

Рисунок 3 – Схема вычисления двумерного преобразования Фурье по строчкам и столбцам

Распараллеливание двумерного преобразования Фурье выполняется на основе указанных выше пунктов.

1.2 Актуальность задачи

Одним из сопутствующих факторов компьютерной революции оказалось появление совершенно новых областей исследования. С каждым годом по мере увеличения быстродействия, уменьшения стоимости и размеров ИС растут возможности решения задач все возрастающей сложности. К ним относится цифровая обработка многомерных сигналов, требующая значительных объемов цифровой памяти и соответствующего количества арифметических операций. Несмотря на сложность, цифровая обработка сигналов уже позволила найти решение ряда важных задач, начиная с компьютерной томографии (методики, позволяющей по проекциям рентгеновского изображения, полученным при различных ориентациях детекторов, выполнять трехмерную реконструкцию органов человеческого тела) и кончая проектированием полей пассивных акустических датчиков и исследованием ресурсов Земли с помощью спутников. Цифровая обработка многомерных сигналов имеет также прочное математическое обоснование, позволяющее не только понять уже достигнутое, но и эффективно исследовать новые проблемы по мере их возникновения и успешно их решать.

По мере развития компьютерной техники меняются требования к обрабатываемым на ней задачам. За последние десять лет компьютеры заметно увеличили количество и уменьшили себестоимость памяти (в частности, оперативной), поэтому в настоящее время возникает потребность в пересмотре уже существующих алгоритмов цифровой обработки данных, например, алгоритмов быстрого двумерного преобразования Фурье, а также их эффективное распараллеливание.

2 Вводная глава

2.1 Точная постановка задачи

Рассмотреть существующие методы вычисления двумерного быстрого преобразования Фурье, определить их достоинства и недостатки, обосновать выбор средств разработки, сформулировать требования к разрабатываемой системе.

Разработать алгоритм распараллеливания для двумерного быстрого преобразования Фурье по строчкам и столбцам и по аналогу Кули-Тьюки для распределенной системы с использованием библиотеки MPI, определить формат хранения данных и промежуточных результатов, способ выделения памяти и способ передачи данных между распределёнными узлами.

Реализовать разработанный алгоритм и провести численный эксперимент. Разработать алгоритм вычисления трёхмерного БПФ по аналогу Кули-Тьюки.

2.2 Преобразование Фурье

Фурье, французский инженер и математик, установил, что любую функцию f(x,y) удовлетворяющую определенным условиям, можно представить в виде суммы бесконечного числа синусоидальных слагаемых. Дирихле в 1829 г. сформулировал условия, при которых имеет место это разложение. Кратко условия Дирихле формулируются следующим образом: $f(t)$ должна быть периодической,

$$f(t) = f(t + 2\pi) \tag{2.1}$$

где 2π – период.

Если f(e) – не периодическая функция, но определена на конечном интервале, то сумма синусоидальных членов все же будет сходиться к f(e) на заданном интервале. Вне этого интервала сумма будет повторением f(e).

f(e) должна быть по меньшей мере кусочно-непрерывной, причем число разрывов конечно, а скачки – ограниченные,

f(e) должна иметь конечное число максимумов и минимумов.

Интеграл

$$\int_{-\pi}^{\pi} f(t)\,dt \tag{2.2}$$

должен сходиться, что следует из условия 2.

В общем случае, согласно теореме Фурье функция f(e), имеющая видимый период T и удовлетворяющая условиям Дирихле, может быть представлена следующим бесконечным рядом Фурье:

$$(t) = a_0 + \sum_{n=1}^{\infty}\left(a_n \cos\frac{2n\pi t}{T} + b_n \sin\frac{2n\pi t}{T}\right), \tag{2.3}$$

где коэффициенты a_0, a_n, b_n рассчитываются по формулам

$$a_0 = \frac{1}{T}\int_{-T/2}^{T/2} f(t)dt, \quad a_n = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\cos\frac{2n\pi t}{T}dt,$$

$$b_n = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\sin\frac{2n\pi t}{T}dt. \tag{2.4}$$

Если $f(t)$ является четной функцией, то все b_n будут равны нулю и в формуле ряда Фурье будут присутствовать только косинусные слагаемые. Если $f(t)$ является нечетной функцией, равны нулю будут, наоборот, косинусные коэффициенты a_n и в формуле останутся лишь синусные слагаемые.

Признак Дини сходимости ряда Фурье в точке.

Пусть функция T - 2π – периодическая и абсолютно интегрируема на периоде. Тогда если x является точкой непрерывности или точкой разрыва первого рода функции f и интеграл

$$\int_0^{\pi} \frac{f_x^*(t)}{t} dt \tag{2.5}$$

абсолютно сходится,

$$\int_0^{\pi} \frac{|f_x^*(t)|}{t} dt < +\infty, \tag{2.6}$$

то ряд Фурье функции f сходится в точке x к значению

$$\frac{f(x+0)+f(x-0)}{2}, \tag{2.7}$$

в частности, в точке непрерывности – к значению f(x) функции f в этой точке [9].

2.3 Двумерное дискретное преобразование Фурье и его обращение

Дискретное прямое и обратное преобразование Фурье допускают непосредственно обобщение и на двумерный случай. Прямое дискретное Фурье-преобразование функции $f(x,y)$ (изображения размерами $M \times N$ задается равенством

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-i2\pi(ux/M+vy/N)}. \qquad (2.8)$$

Как и в одномерном случае, это выражение должно быть вычислено для всех $u = 0,1,2,\dots,M-1$ и также для всех $v = 0,1,2,\dots,N-1$. Аналогично, по заданному Фурье-преобразованию $F(u,v)$, мы можем получить $f(x,y)$ при помощи обратного преобразования Фурье, задаваемого выражением

$$f(x,y) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} F(u,v)e^{-i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}. \qquad (2.9)$$

где $, x = 0,1,2,\dots,M-1$ и $y = 0,1,2,\dots,N-1$.

Элементы Фурье образа в общем случае являются комплексными величинами. Как и в случае комплексных чисел, значение $F(u,v)$ удобно иногда выражать в полярных координатах:

$$F(u,v) = |F(u,v)|e^{-i\phi(u,v)}. \qquad (2.10)$$

где величина

$$|F(u,v)| = [R^2(u,v) + I^2(u,v)]^{1/2} \tag{2.11}$$

называется модулем или спектром Фурье-преобразования, а величина

$$\phi(u,v) = \mathrm{arctg}\left[\frac{I(u,v)}{R(u,v)}\right] \tag{2.12}$$

называются фазой или фазовым спектром преобразования.

Обычной практикой стало умножение исходной функции (изображения) на $(-1)^{x+y}$ перед вычислением фурье-преобразования. Используя свойство экспонент, нетрудно показать, что

$$\mathfrak{F}[f(x,y)(-1)^{x+y}] = F\left(u - \frac{M}{2}, v - \frac{N}{2}\right), \tag{2.13}$$

где $\mathfrak{F}[\cdot]$ обозначает преобразование Фурье своего аргумента. Это равенство означает, что начало координат для Фурье-преобразования функции $f(x,y)(-1)^{x+y}$ (т.е. та точка, где значение этого преобразования равно F(0,0)) находится в точке с координатами $u = M/2, v = N/2$. Другими словами, умножение функции $f(x,y)$ на величину $(-1)^{x+y}$ приводит к сдвигу начала координат для ее образа $F(u,v)$ в точку с частотными координатами $(M/2, N/2)$ которая является центром прямоугольной области размером $(M \times N)$, занимаемой двумерным дискретным Фурье-преобразованием. Мы будем говорить об этой области в частотном пространстве как о частотном прямоугольнике. Он простирается от $u = 0$ до

$u = M - 1$ и от $v = 0$ до $v = N - 1$ (переменные u и v целые). Для того чтобы обеспечить целочисленность координат после сдвига, необходимо, чтобы числа M и N были четными. При компьютерной реализации преобразования Фурье, индексы принимают значения от $u = 1$ до $u = M$ и от $v = 1$ до $v = N$. Подлинный центр Фурье-образа тогда находится в точке

$$u = (M/2) + 1,\ v = (N/2) + 1. \tag{2.14}$$

Значение преобразования в точке $(u,v) = (0,0)$ равно

$$F(0,0) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y) \tag{2.15}$$

т.е. среднему значению функции f(x,y). Другими словами, если f(x,y) изображение, то значение фурье-преобразования в начале координат равно среднему значению яркости на изображении. Поскольку началу координат отвечают нулевые частоты, величину F(0,0) иногда называют постоянной (dc) составляющей спектра. Эта терминология происходит из электротехники, где «dc» обозначает постоянный ток (т.е. ток нулевой частоты).

Если функция f(x,y) вещественная, то ее Фурье-преобразование обладает симметрией по отношению к операции комплексного сопряжения

$$F(u,v) = F^*(-u,-v), \tag{2.16}$$

где «*» означает обычное комплексное сопряжение. Отсюда следует равенство

$$|F(u,v)| = |F(-u,-v)|, \tag{2.17}$$

которое говорит о том, что спектр Фурье-преобразования симметричен.

2.4 Вычисление двумерного дискретного преобразования Фурье

Рассмотрим сигнал f, который является двумерным периодическим сигналом с периодом 2^s по первой и второй координате. Отсчёты задаются, как $f_{k,t}$, где $k, t = 0{:}2^s$.

Дискретное преобразование Фурье для данного сигнала f задаётся формулой

$$F_{l,m} = \sum_{k=0}^{2^s-1} \sum_{t=0}^{2^{s+1}-1} f_{k,t} e^{\frac{2\pi ilk}{2^s}} e^{\frac{2\pi imt}{2^s}}. \tag{2.18}$$

Часто для вычисления дискретного преобразования Фурье (ДПФ) используют алгоритм с применением одномерных дискретных преобразований Фурье – двумерное быстрое преобразование Фурье по строкам и столбцам. Для этого вычисляют F в следующем виде

$$F_{l,m} = \sum_{k=0}^{2^s-1} \left[\sum_{t=0}^{2^s-1} f_{k,t} e^{\frac{2\pi imt}{2^s+1}} \right] e^{\frac{2\pi ilk}{2^s}}. \tag{2.19}$$

Суммы в квадратных скобках представляют собой одномерные вычисления ДПФ по строкам исходного сигнала f. Рассмотрим данный подход с алгоритмом двумерного быстрого преобразования Фурье по аналогу Кули-Тьюки, предложенным ниже.

Во-первых, преобразуем формулу нахождения $F_{l,m}$ следующим образом:

$$F_{l,m} = \sum_{k=0}^{2^s-1}\sum_{t=0}^{2^s-1} f_{k,t} e^{\frac{2\pi i(lk2+mt)}{2^s}} = \sum_{k=0}^{2^s-1}\sum_{t_1=0}^{2^{s-1}-1} f_{k,2t_1} e^{\frac{2\pi i(lk2+m2t_1)}{2^s}} +$$

$$+\sum_{k=0}^{2^s-1}\sum_{t_1=0}^{2^{s-1}-1} f_{k,2(t_1+1)} e^{\frac{2\pi i\left(lk2+m2(t_1+1)\right)}{2^s}} = \sum_{k=0}^{2^s-1}\sum_{t_1=0}^{2^{s-1}-1} f_{k,2t_1} e^{\frac{2\pi i(lk2+m2t_1)}{2^s}} +$$

$$+\sum_{k=0}^{2^s-1}\sum_{t_1=0}^{2^{s-1}-1} f_{k,2(t_1+1)} e^{\frac{2\pi i\left(lk2+m(2t_1+1)\right)}{2^s}} = \sum_{k=0}^{2^s-1}\sum_{t=0}^{2^{s-1}-1} f_{k,2t} e^{\frac{2\pi i(lk+mt)}{2^s}} +$$

$$= \sum_{k=0}^{2^s-1}\sum_{t=0}^{2^{s-1}-1} f_{k,2(t+1)} e^{\frac{2\pi i(lk+mt)}{2^s}} e^{\frac{\pi i m}{2^s}}, \qquad (2.20)$$

где $l, m = 0{:}\,2^s - 1$.

Можно показать, что $e^{\frac{\pi i m}{2^s}}$ обладает свойством симметрии относительно $m = 2^{s-1}$.

$$e^{\frac{\pi i(2^s+t)}{2^s}} = e^{\frac{\pi i 2^s}{2^s}} \cdot e^{\frac{\pi i t}{2^s}} = -e^{\frac{\pi i t}{2^s}}, \qquad (2.21)$$

где $t = 0{:}\,2^{s-1} - 1$.

Сумма

$$\sum_{k=0}^{2^s-1}\sum_{t=0}^{2^{s-1}-1} f_{k,2t} e^{\frac{2\pi i(lk+mt)}{2^s}} \qquad (2.22)$$

является двумерным ДПФ сигнала $\widetilde{f_{k,t}} = f_{k,2t}$, то есть сигнала, полученного прореживанием исходного сигнала $f_{k,t}$ по чётным отсчётам второй переменной; обозначим данной ДПФ через $F'_{l,m}$. Аналогично видно, что сумма

$$\sum_{k=0}^{2^s-1} \sum_{t=0}^{2^{s-1}-1} f_{k,2t+1} e^{\frac{2\pi i(lk+mt)}{2^s}} \tag{2.23}$$

является ДПФ сигнала $\widetilde{\widetilde{f_{k,t}}} = f_{k,2t+1}$, полученного из исходного сигнала прореживание по нечётным отсчётам второй переменной; обозначим данное ДПФ, как $F''_{l,m}$.

Из (2.20) и (2.21) и введённых выше обозначений получаем, что

$$F_{l,m} = F'_{l,m} + e^{\frac{\pi i m}{2^s}} F''_{l,m},$$
$$F_{l,2^s+m} = F'_{l,m} - e^{\frac{\pi i m}{2^s}} F''_{l,m}, \tag{2.24}$$

где $l, m \in 0: 2^s - 1$.

Таким образом, нахождение двумерного дискретного преобразования Фурье сигнала $f_{k,t}$ можно реализовать по следующему алгоритму:

1) Нахождение спектра F' сигнала $\widetilde{f}$, где $\widetilde{f_{k,t}} = f_{k,2t}$.

2) Нахождение спектра F'' $\widetilde{\widetilde{f}}$, где $\widetilde{\widetilde{f_{k,t}}} = f_{k,2t+1}$.

3) Домножение отсчётов спектра $F''_{l,m}$ на дополнительный множитель $e^{\frac{\pi i m}{2^s}}$.

4) Определение значений спектра сигнала f, выражение (2.24).

Рассмотрим количество комплексных сложений (вычитаний) и умножений в приведенном выше алгоритме. Для сигналов $\tilde{f}$ и $\tilde{\tilde{f}}$ отсчеты спектров F' и F'' могут вычислены при помощи $\frac{3}{4}2^{2s}s$ комплексных умножений и $2 \cdot 2^{2s}s$ комплексных сложений (вычитаний).

Данные оценки могут быть доказаны распространением теории об рекуррентных последовательностях в пространстве сигналов на двумерный случай. Отметим, что для одномерного сигнала, где количество отсчетов $N = 2^s$ является степенью двойки, количество комплексных сложений равно $2^s s$, а количество комплексных умножений $\frac{1}{2}2^s s$.

Тогда из алгоритма (4) нахождение спектра сигнала $f_{k,t}$ $(k = 0{:}2^s - 1, t = 0{:}2^{s+1} - 1)$ потребует

$$\frac{3}{4}2^{2s}s + 2^s \cdot 2^s = \frac{3}{4}2^{2s}s + 2^{2s} \tag{2.25}$$

комплексных умножений и

$$2^s 2^s s + 2^s 2^s (s+1) = 2 \cdot 2^{2s}s + 2^{2s} \tag{2.26}$$

комплексных сложений.

Для вычисления алгоритма двумерного быстрого преобразования Фурье по строчкам и столбцам потребуется $2^{3s}s$ комплексных сложений и $\frac{1}{2}2^{3s}s$ комплексных умножений.

Отсюда можно сделать вывод, что применение алгоритма (4) более эффективно, чем алгоритма, использующего нахождение одномерных ДПФ, поскольку происходит уменьшение количества операций комплексного умножения при сохранении количества операций комплексного сложения.

Теперь рассмотрим сигнал f, который является двумерным периодическим сигналом с периодом 2^s по первой и 2^{s+1} по второй координате. Отсчёты задаются, как $f_{k,t}$, где $k = 0:2^s, t = 0:2^{s+1}$.

Дискретное преобразование Фурье для данного сигнала f задаётся формулой

$$F_{l,m} = \sum_{k=0}^{2^s-1} \sum_{t=0}^{2^{s+1}-1} f_{k,t} e^{\frac{2\pi ilk}{2^s}} e^{\frac{2\pi imt}{2^{s+1}}}. \tag{2.27}$$

Тогда формула для двумерного быстрого преобразования Фурье по строчкам и столбцам примет вид:

$$F_{l,m} = \sum_{k=0}^{2^s-1} \left[\sum_{t=0}^{2^{s+1}-1} f_{k,t} e^{\frac{2\pi imt}{2^s+1}} \right] e^{\frac{2\pi ilk}{2^s}}. \tag{2.28}$$

По аналогии можно получить формулу для двумерного быстрого преобразования Фурье по аналогу Кули-Тьюки:

$$F_{l,m} = \sum_{k=0}^{2^s-1} \sum_{t=0}^{2^{s+1}-1} f_{k,t} e^{\frac{2\pi i(lk2+mt)}{2^{s+1}}} \sum_{k=0}^{2^s-1} \sum_{t_1=0}^{2^s-1} f_{k,2t_1} e^{\frac{2\pi i(lk2+m2t_1)}{2^{s+1}}} +$$

$$+ \sum_{k=0}^{2^s-1} \sum_{t_1=0}^{2^s-1} f_{k,2(t_1+1)} e^{\frac{2\pi i(lk2+m2(t_1+1))}{2^{s+1}}} = \sum_{k=0}^{2^s-1} \sum_{t_1=0}^{2^s-1} f_{k,2t_1} e^{\frac{2\pi i(lk2+m2t_1)}{2^{s+1}}} +$$

$$+ \sum_{k=0}^{2^s-1} \sum_{t_1=0}^{2^s-1} f_{k,2(t_1+1)} e^{\frac{2\pi i(lk2+m(2t_1+1))}{2^{s+1}}} = \sum_{k=0}^{2^s-1} \sum_{t=0}^{2^s-1} f_{k,2t} e^{\frac{2\pi i(lk+mt)}{2^s}} +$$

$$+ \sum_{k=0}^{2^s-1} \sum_{t=0}^{2^s-1} f_{k,2(t+1)} e^{\frac{2\pi i(lk+mt)}{2^s}} e^{\frac{\pi im}{2^s}}, \tag{2.29}$$

где $l = 0{:}\,2^s - 1, m = 0{:}\,2^{s+1} - 1$.

Можно показать, что $e^{\frac{\pi i m}{2^s}}$ обладает свойством симметрии относительно $m = 2^s$.

$$e^{\frac{\pi i(2^s+t)}{2^s}} = e^{\frac{\pi i 2^s}{2^s}} \cdot e^{\frac{\pi i t}{2^s}} = -e^{\frac{\pi i t}{2^s}}, \tag{2.30}$$

где $t = 0{:}\,2^s - 1$.

Сумма

$$\sum_{k=0}^{2^s-1} \sum_{t=0}^{2^s-1} f_{k,2t} e^{\frac{2\pi i(lk+mt)}{2^s}} \tag{2.31}$$

является двумерным ДПФ сигнала $\widetilde{f_{k,t}} = f_{k,2t}$, то есть сигнала, полученного прореживанием исходного сигнала $f_{k,t}$ по чётным отсчётам второй переменной; обозначим данной ДПФ через $F'_{l,m}$. Аналогично видно, что сумма

$$\sum_{k=0}^{2^s-1} \sum_{t=0}^{2^s-1} f_{k,2t+1} e^{\frac{2\pi i(lk+mt)}{2^s}} \tag{2.32}$$

является ДПФ сигнала $\widetilde{\widetilde{f_{k,t}}} = f_{k,2t+1}$, полученного из исходного сигнала прореживание по нечётным отсчётам второй переменной; обозначим данное ДПФ, как $F''_{l,m}$.

Из (2.29) и (2.30) и введённых выше обозначений получаем, что

$$F_{l,m} = F'_{l,m} + e^{\frac{\pi i m}{2^s}} F''_{l,m},$$

$$F_{l,2^s+m} = F'_{l,m} - e^{\frac{\pi i m}{2^s}} F''_{l,m}, \qquad (2.33)$$

где $l, m \in 0: 2^s - 1$.

Таким образом, нахождение двумерного дискретного преобразования Фурье сигнала $f_{k,t}$ можно реализовать по следующему алгоритму:

1) Нахождение спектра F' сигнала $\tilde{f}$, где $\widetilde{f_{k,t}} = f_{k,2t}$

2) Нахождение спектра F'' $\tilde{\tilde{f}}$, где $\widetilde{\widetilde{f_{k,t}}} = f_{k,2t+1}$

3) Домножение отсчётов спектра $F''_{l,m}$ на дополнительный множитель $e^{\frac{\pi i m}{2^s}}$.

4) Определение значений спектра сигнала f, выражение (2.33).

Рассмотрим количество комплексных сложений (вычитаний) и умножений в приведенном выше алгоритме. Для сигналов $\tilde{f}$ и $\tilde{\tilde{f}}$ отсчеты спектров F' и F'' могут вычислены при помощи $\frac{3}{4} 2^{2s} s$ комплексных умножений и $2 \cdot 2^{2s} s$ комплексных сложений (вычитаний).

Данные оценки могут быть доказаны распространением теории об рекуррентных последовательностях в пространстве сигналов на двумерный случай. Отметим, что для одномерного сигнала, где количество отсчетов $N = 2^s$ является степенью двойки, количество комплексных сложений равно $2^s s$, а количество комплексных умножений $\frac{1}{2} 2^s s$.

Тогда из алгоритма нахождение спектра сигнала

$f_{k,t}$ $(k = 0: 2^s - 1, t = 0: 2^{s+1} - 1)$ потребует

$$2 \cdot \frac{3}{4} 2^{2s} s + 2^s \cdot 2^s = \frac{3}{2} 2^{2s} s + 2^{2s} \qquad (2.34)$$

комплексных умножений и

$$2^{s+1}2^{s}s + 2^{s}2^{s+1}(s+1) = 4 \cdot 2^{2s}s + 2^{2s+1} \qquad (2.35)$$

комплексных сложений.

Отсюда можно сделать вывод, что, как в прошлой ситуации, применение алгоритма двумерного быстрого преобразования Фурье по аналогу Кули-Тьюки более эффективно, чем алгоритма быстрого преобразования Фурье по строчкам и столбцам, так как происходит уменьшение количества операций комплексного умножения при сохранении количества операций комплексного сложения.

2.5 Вычисление трёхмерного дискретного преобразования Фурье

Рассмотрим сигнал f, который является трехмерным периодическим сигналом с периодом 2^s по трем координатам. Отсчёты задаются, как $f(x, y, z)$, где $x, y, z = 0 : 2^s$. Дискретное преобразование Фурье для данного сигнала f задаётся формулой:

$$F(a, b, c) = \sum_{x=0}^{2^s-1} \sum_{y=0}^{2^s-1} \sum_{z=0}^{2^s-1} f(x, y, z) \cdot e^{\frac{2\pi i (ax+by+cz)}{2^s}} \qquad (2.36)$$

Трехмерное ДПФ Фурье F можно вычислить при помощи одномерных:

$$F(a, b, c) = \sum_{x=0}^{2^s-1} \left[\sum_{y=0}^{2^s-1} \left[\sum_{z=0}^{2^s-1} f(x, y, z) \cdot e^{\frac{2\pi i a x}{2^s}} \right] \cdot e^{\frac{2\pi i b y}{2^s}} \right] \cdot e^{\frac{2\pi i c z}{2^s}} \qquad (2.37)$$

Суммы в квадратных скобках представляют собой одномерные вычисления ДПФ по трём координатам сигнала f. Преобразуем данную формулу разбиением трех координат на четную и нечетную компоненты:

$$F(a, b, c) = \sum_{x=0}^{2^s-1} \sum_{y=0}^{2^s-1} \sum_{z=0}^{2^s-1} f(x, y, z) e^{\frac{2\pi i a x}{2^s}} e^{\frac{2\pi i b y}{2^s}} e^{\frac{2\pi i c z}{2^s}} = \qquad (2.38)$$

$$= \sum_{x_1=0}^{2^{s-1}-1}\sum_{y_1=0}^{2^{s-1}-1}\sum_{z_1=0}^{2^{s-1}-1} f(2x_1,2y_1,2z_1)\, e^{\frac{2\pi i a x_1}{2^{s-1}}} e^{\frac{2\pi i b y_1}{2^{s-1}}} e^{\frac{2\pi i c z_1}{2^{s-1}}} +$$

$$+ e^{\frac{\pi i a}{2^s}} \sum_{x_1=0}^{2^{s-1}-1}\sum_{y_1=0}^{2^{s-1}-1}\sum_{z_1=0}^{2^{s-1}-1} f(2x_1+1,2y_1,2z_1)\, e^{\frac{2\pi i a x_1}{2^{s-1}}} e^{\frac{2\pi i b y_1}{2^{s-1}}} e^{\frac{2\pi i c z_1}{2^{s-1}}} +$$

$$+ e^{\frac{\pi i b}{2^s}} \sum_{x_1=0}^{2^{s-1}-1}\sum_{y_1=0}^{2^{s-1}-1}\sum_{z_1=0}^{2^{s-1}-1} f(2x_1,2y_1+1,2z_1)\, e^{\frac{2\pi i a x_1}{2^{s-1}}} e^{\frac{2\pi i b y_1}{2^{s-1}}} e^{\frac{2\pi i c z_1}{2^{s-1}}} +$$

$$+ e^{\frac{\pi i a}{2^s}} e^{\frac{1\pi i b}{2^s}} \sum_{x_1=0}^{2^{s-1}-1}\sum_{y_1=0}^{2^{s-1}-1}\sum_{z_1=0}^{2^{s-1}-1} f(2x_1+1,2y_1+1,2z_1)\, e^{\frac{2\pi i a x_1}{2^{s-1}}} e^{\frac{2\pi i b y_1}{2^{s-1}}} e^{\frac{2\pi i c z_1}{2^{s-1}}} +$$

$$+ e^{\frac{\pi i c}{2^s}} \sum_{x_1=0}^{2^{s-1}-1}\sum_{y_1=0}^{2^{s-1}-1}\sum_{z_1=0}^{2^{s-1}-1} f(2x_1,2y_1,2z_1+1)\, e^{\frac{2\pi i a x_1}{2^{s-1}}} e^{\frac{2\pi i b y_1}{2^{s-1}}} e^{\frac{2\pi i c z_1}{2^{s-1}}} +$$

$$+ e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i c}{2^s}} \sum_{x_1=0}^{2^{s-1}-1}\sum_{y_1=0}^{2^{s-1}-1}\sum_{z_1=0}^{2^{s-1}-1} f(2x_1+1,2y_1,2z_1+1)\, e^{\frac{2\pi i a x_1}{2^{s-1}}} e^{\frac{2\pi i b y_1}{2^{s-1}}} e^{\frac{2\pi i c z_1}{2^{s-1}}} +$$

$$+ e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} \sum_{x_1=0}^{2^{s-1}-1}\sum_{y_1=0}^{2^{s-1}-1}\sum_{z_1=0}^{2^{s-1}-1} f(2x_1,2y_1+1,2z_1+1)\, e^{\frac{2\pi i a x_1}{2^{s-1}}} e^{\frac{2\pi i b y_1}{2^{s-1}}} e^{\frac{2\pi i c z_1}{2^{s-1}}} +$$

$$+ e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} \sum_{x_1=0}^{2^{s-1}-1}\sum_{y_1=0}^{2^{s-1}-1}\sum_{z_1=0}^{2^{s-1}-1} f(2x_1+1,2y_1+1,2z_1+1)\, e^{\frac{2\pi i a x_1}{2^{s-1}}} e^{\frac{2\pi i b y_1}{2^{s-1}}} e^{\frac{2\pi i c z_1}{2^{s-1}}} =$$

$$= F_{0,0,0}(a,b,c) + e^{\frac{\pi i a}{2^s}} F_{1,0,0}(a,b,c) + e^{\frac{\pi i b}{2^s}} F_{0,1,0}(a,b,c) + e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} F_{1,1,0}(a,b,c) +$$

$$+ e^{\frac{\pi i c}{2^s}} F_{0,0,1}(a,b,c) + e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,0,1}(a,b,c) + e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{0,1,1}(a,b,c) + e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,1,1}(a,b,c),$$

где $a,b,c = 0:2^s-1$.

Можно показать, что $e^{\frac{\pi i a}{2^s}}$ обладает свойством симметрии относительно $a = 2^{s-1}$:

$$e^{\frac{\pi i (2^{s-1}+t)}{2^s}} = e^{\frac{\pi i 2^{s-1}}{2^s}} \cdot e^{\frac{\pi i t}{2^s}} = -e^{\frac{\pi i t}{2^s}} \qquad (2.39)$$

где $t = 0:2^{s-1}-1$. Аналогично, $e^{\frac{\pi i b}{2^s}}$ и $e^{\frac{\pi i c}{2^s}}$ обладают симметрией относительно $b = 2^{s-1}$ и $c = 2^{s-1}$, соответственно. Тогда из (3) и (4) получим:

$$F(a,b,c) = F_{0,0,0}(a,b,c) + e^{\frac{\pi i a}{2^s}} F_{1,0,0}(a,b,c) + e^{\frac{\pi i b}{2^s}} F_{0,1,0}(a,b,c) + e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} F_{1,1,0}(a,b,c) +$$

$$+ e^{\frac{\pi i c}{2^s}} F_{0,0,1}(a,b,c) + e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,0,1}(a,b,c) + e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{0,1,1}(a,b,c) + e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,1,1}(a,b,c), \qquad (2.40)$$

$$F(a+2^{s-1},b,c) = F_{0,0,0}(a+2^{s-1},b,c) - e^{\frac{\pi i a}{2^s}} F_{1,0,0}(a+2^{s-1},b,c) + e^{\frac{\pi i b}{2^s}} F_{0,1,0}(a+2^{s-1},b,c) -$$

$$-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} F_{1,1,0}\left(a+2^{s-1},b,c\right)+e^{\frac{\pi i c}{2^s}} F_{0,0,1}\left(a+2^{s-1},b,c\right)-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,0,1}\left(a+2^{s-1},b,c\right)+$$
$$+e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{0,1,1}\left(a+2^{s-1},b,c\right)-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,1,1}\left(a+2^{s-1},b,c\right),$$

$$F\left(a,b+2^{s-1},c\right)=F_{0,0,0}\left(a,b+2^{s-1},c\right)+e^{\frac{\pi i a}{2^s}} F_{1,0,0}\left(a,b+2^{s-1},c\right)-e^{\frac{\pi i b}{2^s}} F_{0,1,0}\left(a,b+2^{s-1},c\right)-$$
$$-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} F_{1,1,0}\left(a,b+2^{s-1},c\right)+e^{\frac{\pi i c}{2^s}} F_{0,0,1}\left(a,b+2^{s-1},c\right)+e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,0,1}\left(a,b+2^{s-1},c\right)-$$
$$-e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{0,1,1}\left(a,b+2^{s-1},c\right)-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,1,1}\left(a,b+2^{s-1},c\right),$$

$$F\left(a,b,c+2^{s-1}\right)=F_{0,0,0}\left(a,b,c+2^{s-1}\right)+e^{\frac{\pi i a}{2^s}} F_{1,0,0}\left(a,b,c+2^{s-1}\right)+e^{\frac{\pi i b}{2^s}} F_{0,1,0}\left(a,b,c+2^{s-1}\right)+$$
$$+e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} F_{1,1,0}\left(a,b,c+2^{s-1}\right)-e^{\frac{\pi i c}{2^s}} F_{0,0,1}\left(a,b,c+2^{s-1}\right)-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,0,1}\left(a,b,c+2^{s-1}\right)-$$
$$-e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{0,1,1}\left(a,b,c+2^{s-1}\right)-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,1,1}\left(a,b,c+2^{s-1}\right),$$

$$F\left(a+2^{s-1},b+2^{s-1},c\right)=F_{0,0,0}\left(a+2^{s-1},b+2^{s-1},c\right)-e^{\frac{\pi i a}{2^s}} F_{1,0,0}\left(a+2^{s-1},b+2^{s-1},c\right)-$$
$$-e^{\frac{\pi i b}{2^s}} F_{0,1,0}\left(a+2^{s-1},b+2^{s-1},c\right)+e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} F_{1,1,0}\left(a+2^{s-1},b+2^{s-1},c\right)+$$
$$+e^{\frac{\pi i c}{2^s}} F_{0,0,1}\left(a+2^{s-1},b+2^{s-1},c\right)-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,0,1}\left(a+2^{s-1},b+2^{s-1},c\right)-$$
$$-e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{0,1,1}\left(a+2^{s-1},b+2^{s-1},c\right)+e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,1,1}\left(a+2^{s-1},b+2^{s-1},c\right),$$

$$F\left(a+2^{s-1},b,c+2^{s-1}\right)=F_{0,0,0}\left(a+2^{s-1},b,c+2^{s-1}\right)-e^{\frac{\pi i a}{2^s}} F_{1,0,0}\left(a+2^{s-1},b,c+2^{s-1}\right)+$$
$$+e^{\frac{\pi i b}{2^s}} F_{0,1,0}\left(a+2^{s-1},b,c+2^{s-1}\right)-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} F_{1,1,0}\left(a+2^{s-1},b,c+2^{s-1}\right)-$$
$$-e^{\frac{\pi i c}{2^s}} F_{0,0,1}\left(a+2^{s-1},b,c+2^{s-1}\right)+e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,0,1}\left(a+2^{s-1},b,c+2^{s-1}\right)-$$
$$-e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{0,1,1}\left(a+2^{s-1},b,c+2^{s-1}\right)+e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,1,1}\left(a+2^{s-1},b,c+2^{s-1}\right),$$

$$F\left(a,b+2^{s-1},c+2^{s-1}\right)=F_{0,0,0}\left(a,b+2^{s-1},c+2^{s-1}\right)+e^{\frac{\pi i a}{2^s}} F_{1,0,0}\left(a,b+2^{s-1},c+2^{s-1}\right)-$$
$$-e^{\frac{\pi i b}{2^s}} F_{0,1,0}\left(a,b+2^{s-1},c+2^{s-1}\right)-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} F_{1,1,0}\left(a,b+2^{s-1},c+2^{s-1}\right)-$$
$$-e^{\frac{\pi i c}{2^s}} F_{0,0,1}\left(a,b+2^{s-1},c+2^{s-1}\right)-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,0,1}\left(a,b+2^{s-1},c+2^{s-1}\right)+$$
$$+e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{0,1,1}\left(a,b+2^{s-1},c+2^{s-1}\right)+e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,1,1}\left(a,b+2^{s-1},c+2^{s-1}\right),$$

$$F\left(a+2^{s-1},b+2^{s-1},c+2^{s-1}\right)=F_{0,0,0}\left(a+2^{s-1},b+2^{s-1},c+2^{s-1}\right)-$$
$$-e^{\frac{\pi i a}{2^s}} F_{1,0,0}\left(a+2^{s-1},b+2^{s-1},c+2^{s-1}\right)-e^{\frac{\pi i b}{2^s}} F_{0,1,0}\left(a+2^{s-1},b+2^{s-1},c+2^{s-1}\right)+$$
$$+e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} F_{1,1,0}\left(a+2^{s-1},b+2^{s-1},c+2^{s-1}\right)-e^{\frac{\pi i c}{2^s}} F_{0,0,1}\left(a+2^{s-1},b+2^{s-1},c+2^{s-1}\right)+$$
$$+e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,0,1}\left(a+2^{s-1},b+2^{s-1},c+2^{s-1}\right)+e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{0,1,1}\left(a+2^{s-1},b+2^{s-1},c+2^{s-1}\right)-$$

$$-e^{\frac{\pi i a}{2^s}} e^{\frac{\pi i b}{2^s}} e^{\frac{\pi i c}{2^s}} F_{1,1,1}\left(a+2^{s-1}, b+2^{s-1}, c+2^{s-1}\right),$$

где a,b.c = 0 ÷ 2^{s-1} - 1.

Число комплексных умножений алгоритма трёхмерного БПФ по аналогу Кули-Тьюки 7/8N^3 $\log_2$ N комплексных сложений 3N^3 $\log_2$ N, в отличие от метода вычисления при помощи комбинации одномерных БПФ: N^3 $\log_2$ N комплексных умножений и N^3 $\log_2$ N комплексных сложений.

2.6 Описание библиотеки MPI

Наиболее распространённой технологией программирования для параллельных компьютеров с распределённой памятью в настоящее время является MPI. Основным способом взаимодействия параллельных процессов в таких системах является передача сообщений друг другу. Это и отражено в названии данной технологии - Message Passing Interface (интерфейс передачи данных). Стандарт MPI фиксирует интерфейс, который должен соблюдаться как системой программирования на каждой вычислительной платформе, так и пользователем при создании своих программ. Современные реализации чаще всего соответствуют стандарту MPI версии 1.1. В 1997-1998 годах появился стандарт MPI-2.0, значительно расширивший функциональность предыдущей версии. Однако до сих пор этот вариант MPI не получил широкого распространения и в полном объёме не реализован не в одной системе.

Интерфейс MPI поддерживает создание параллельных программ в стиле MIMD (Multiple Instruction Multiple Data), что подразумевает объединение процессов с различными исходными текстами. Однако писать и отлаживать такие программы очень сложно, поэтому на практике программисты гораздо чаще используют SPMD - модель (Single Program Multiple Data) параллельного

программирования, в рамках которой для всех параллельных процессов используется один и тот же код.

MPI-программа - это множество параллельных взаимодействующих процессов. Все процессы порождаются один раз, образуя параллельную часть программы. В ходе выполнения MPI - программы порождение дополнительных процессов или уничтожение существующих не допускается (в MPI-2.0 такая возможность появилась). Каждый процесс работает в своём адресном пространстве, никаких общих переменных или данных в MPI нет. Основным способом взаимодействия между процессами является явная посылка сообщений.

Для локализации взаимодействия параллельных процессов программы можно создать группы процессов, предоставляя им отдельную среду для общения – коммуникатор. Состав образуемых групп произволен. Группы могут полностью совпадать, входить одна в другую, не пересекаться или пересекаться частично. Процессы могут взаимодействовать только внутри некоторого коммуникатора, сообщения, отправленные в разных коммуникаторах, не пересекаются и не мешают друг другу. При старте программы всегда считается, что все порождённые процессы работают в рамках всеобъемлющего коммуникатора, имеющего предопределённое имя - MPI_COMM_WPRLD. Этот коммуникатор существует всегда и служит для взаимодействия всех запущенных процессов MPI-программы. Все взаимодействия протекают в разных коммуникаторах, никак не мешают друг другу.

Каждый процесс MPI-программы имеет в каждой группе, в которую он входит, уникальный атрибут – номер процесса, который является целым неотрицательным числом. В одном и том же коммуникаторе все процессы имеют различные номера. Но процесс может входить в другой коммуникатор, он будет иметь там другой номер. Поэтому два основных атрибута процесса:

коммуникатор и номер в коммуникаторе. Если группа содержит n процессов, то номер любого процесса в данной группе лежит в пределах от 0 до n-1.

Основным способом общения процессов между собой является явная посылка сообщений. Сообщение – это набор данных некоторого типа. Каждое сообщение имеет несколько атрибутов, в частности, номер процесса0отправителя, номер процесса-получателя, идентификатор сообщения и другие. Одним из важных атрибутов сообщения является его идентификатор или тэг. По идентификатору процесс, принимающий сообщение, например, может различить два сообщения, пришедшие к нему от одного и того же процесса. Сам идентификатор сообщения является целым неотрицательным числом, лежащим в диапазоне от 0 до MPI_TAG_UP, причём гарантируется, что MPI_TAG_UP не меньше 32767. Для работы с атрибутами сообщений в языке Си введена структура, элементы которой дают доступ к их значениям.

2.7 Комплекс высокопроизводительных вычислений ИКИТ СФУ

В Сибирском Федеральном Университете на базе Института Космических и Информационных технологий развернут мощный комплекс высокопроизводительных вычислений суперкомпьютер ИКИТ СФУ.

Суперкомпьютер ИКИТ СФУ обеспечивает научных сотрудников, преподавателей, аспирантов и студентов современными вычислительными мощностями в проведении исследований и экспериментов. В составе комплекса находятся 224 вычислительных узла IBM Blade HS21. Каждый узел включает в себя 4 Gb оперативной памяти и два четырехядерных процессора Xeon quad core 2.33 GHz; два высокопроизводительных узла IBM System X 3950, объединенные в одну вычислительную систему по четыре четырехядерных процессора и 16 Gb оперативной памяти в каждом узле. Система хранения данных IBM DS3400 представляет собой единое дисковое пространство

емкостью 20 Tb. Пропускная способность вычислительной сети Infiniband, которая объединяет узлы кластера, достигает 20 Gbit/s.

Производительность комплекса, согласно тесту linpack, составляет 13,057 TFlops. Пиковая производительность - 16872,3 MFlops. Суперкомпьютер СФУ, на сегодняшний день, занимает 14 место в списке пятидесяти самых мощных компьютеров России.

Комплекс активно используют в научных исследованиях профессорско-преподавательский состав СФУ и других вузов. Приобретается и создается системное и прикладное программное обеспечение, которое позволяет решать сложные задачи в таких областях, как механика твердого тела, жидкости и газа, геология, гео- и гидрофизика, электродинамика, экология, а так же в задачах, связанных с проектированием технических устройств и систем. Суперкомпьютер работает под управлением операционной системы Linux и выполняет задачи на специализированном, кластерном программном обеспечении OpenMPI.

Комплекс Высокопроизводительных вычислений ИКИТ СФУ обладает наибольшей вычислительной мощностью на территории Сибири. Сам комплекс представляет собой специализированное оборудование и программное обеспечение компании IBM (систему мониторинга и управления комплексом), в сочетании со свободным программным обеспечением. Кластер используется научными сотрудниками для расчетов своих прикладных задач и студентами СФУ в образовательных целях.

Рисунок 4 - Комплекс высокопроизводительных вычислений ИКИТ СФУ

3 Основные достигнутые результаты

3.1 Словесное описание алгоритма работы программы

Программа написана с применением клиент-серверной архитектуры. Роль сервера выполняет главный процесс (с рангом 0). Он осуществляет считывание и запись данных в файл, перестановку элементов сигнала в соответствии с алгоритмом одномерного быстрого преобразования Фурь Кули-Тьюки, пересылает и принимает данные от других процессов, а также замеряет время работы алгоритмов.

Программу можно разбить на несколько смысловых частей:

1) Ввод данных из файла,
2) Двумерное быстрое преобразование Фурье по строчкам и столбцам,
3) Сохранение результата,

4) Ввод данных из файла,

5) Двумерное быстрое преобразование Фурье по аналогу Кули-Тьюки,

6) Сохранение результата.

После начала работы программы все процессы выделяют память под свои локальные переменные (счетчики, буфер для счета, приема и передачи данных), а главный процесс дополнительно выделяет память для считывания и хранения исходного сигнала целиком. Затем главный процесс считывает сигнал из файла, и осуществляет их перестановку, после чего программа выполняет двумерное быстрое преобразование Фурье по строчкам и столбцам следующим образом.

Сначала идет проход по строкам – выполнения одномерного быстрого преобразования Фурье Кули-Тьюки над каждой строкой. При этом главный процесс посылает набор строк каждому процессу (всем одинаковое количество) и остаток оставляет себе. Все процессы выполняют одномерное быстрое преобразование Фурье Кули-Тьюки над своим набором строк и посылают результат главному процессу. После этого главный процесс принимает результат от остальных, транспонирует полученную матрицу и снова отсылает набор строк (столбцов в исходном сигнале) для вычисления. После получения результата главный процесс сохраняет полученный спектр в файл, снова загружает исходные данные, и программа выполняет двумерное быстрое преобразование Фурье по аналогу Кули-Тьюки.

В каждой из рекурсий главный процесс посылает остальным пару взаимосвязанных в данной рекурсии строк. Далее все процессы выполняют шаг рекурсии, оперируя при этом сразу с четверкой чисел (в одномерном случае - с парой чисел) при вычислении двумерного быстрого преобразования Фурье по аналогу Кули-Тьюки. После подсчета все процессы (кроме главного) пересылают ему результат, а главный принимает и переходит к следующему шагу рекурсии. По окончании преобразования Фурье по аналогу Кули-Тьюки главный процесс сохраняет спектр в файл. Перед завершением программы все процессы удаляют память под счетчики и локальные переменные для хранения данных, а главный процесс дополнительно удаляет память для хранения исходного сигнала и его спектра.

3.1 Блок-схема алгоритма работы программы

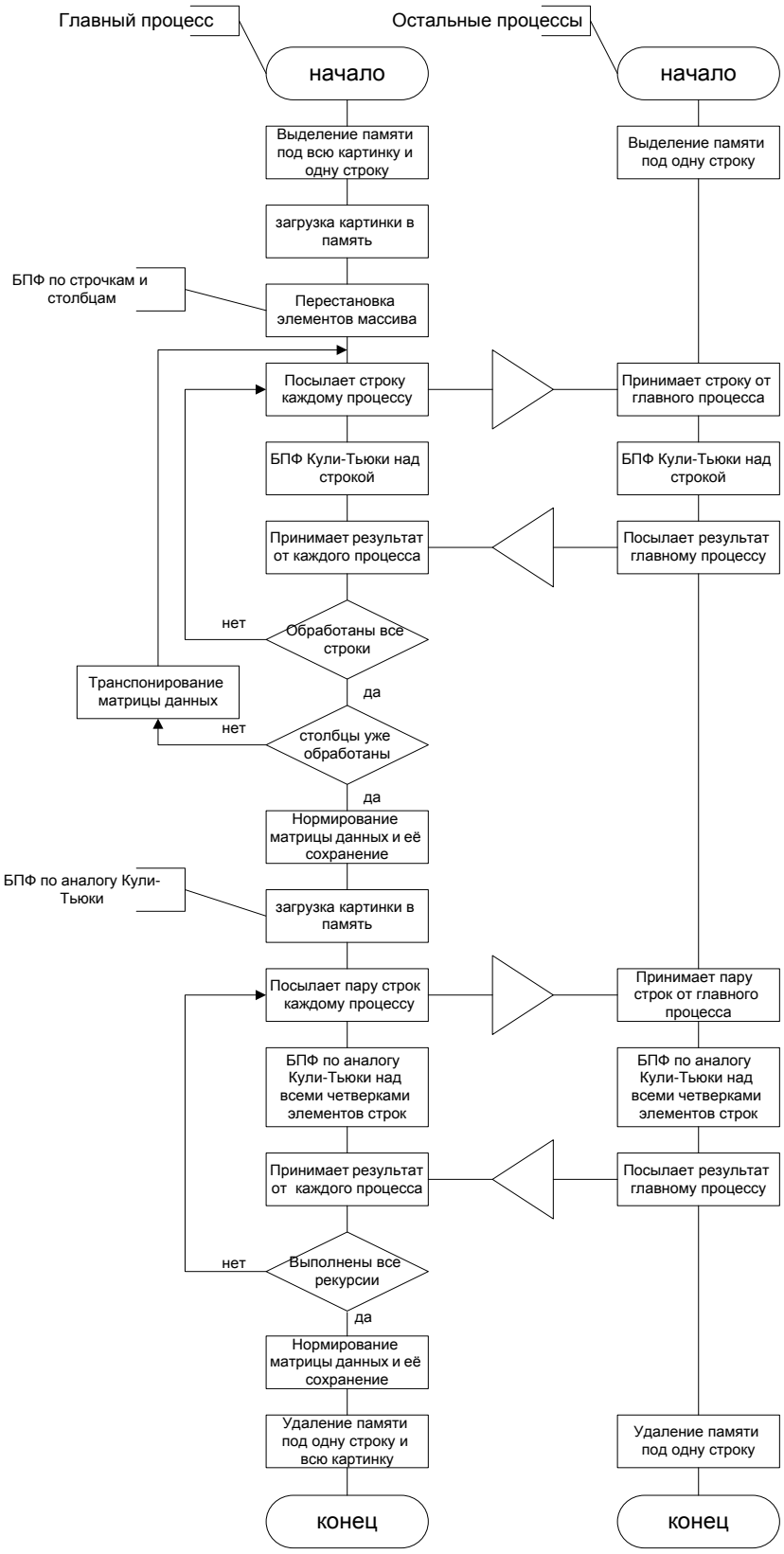

Рисунок 5 – Блок-схема алгоритма работы пропграммы

3.2 Основные трудности, возникшие в процессе работы

В ходе написания программы передо мной возникли следующие трудности:

Способ хранения и обработки комплексных чисел

Первоначальное решение - это использование стандартного класса complex<type>, хранящего пару чисел (вещественную и мнимую части). Основное его преимущество - это готовые операции с комплексными числами (умножение, деление, сложение, вычитание, взятие модуля, сопряжение), а также комплексная экспонента, что заметно упрощает написание программы. Также к преимуществам можно отнести приватность данных (данные можно изменять только через вызов специальных функций, а напрямую нельзя), благодаря чему данные нельзя будет случайно изменить в ходе написания программы. Но это преимущество является и недостатком, так как на вызов функций для изменения данных комплексного числа требуется дополнительное время. Для решения этой проблемы мной было принято решение не использовать класс complex<type>, а хранить реальную и мнимую части числа в двух переменных (массивах) и обращаться к ним напрямую. Однако такой метод заключает в себе некоторые сложности. Одна из них – это отсутствие готовых операций с комплексными числами, их пришлось реализовывать вручную в коде программы в каждом месте, где необходимо сложить (вычесть, умножить, разделить) два комплексных числа (для исключения частых вызовов функций - операций с комплексными числами). Вторая сложность – это отсутствие защиты данных от случайной перезаписи в процессе выполнения/написания кода программы. Обе эти сложности усложнили процесс написания программы, однако такой подход (хранение реальной и мнимой части числа не в структуре, а в двух переменных) увеличил производительность программы в десятки раз.

Выбор типа данных для хранения и обработки двумерного сигнала

Для работы с вещественными числами в языке программирования C++ используются два типа данных: float и double. Тип double - это число с двойной точностью (по сравнению с типом float) , что приводит к большой точности вычислений. К тому же в современных процессорах время выполнения операций умножения и сложения для типов double и float одинаково, что не сказывается на скорости вычислений. Однако для хранения одного числа типа double необходимо выделить 8 байт памяти (например, для сигнала размером 8192×8192 потребуется 512 МГб), для типа float всего 4 байта. Так как использование дискретного входного сигнала заметно снижает точность вычислений, то использование данного типа не скажется на результате. Ещё одним плюсом использования в программе типа данных float является то, что за счет меньшего объёма памяти вероятность попадания в КЭШ-память процессора намного выше, следовательно, происходит меньше обращений к оперативной памяти, что ускоряет работу программы.

Вычисление комплексной экспоненты

Самый простой способ – это прямое вычисление значения экспоненты на каждом шаге. Этот метод имеет наибольшую точность, однако требует больших затрат времени (например, время выполнения операций сложения, умножения вещественных чисел - несколько наносекунд, вычисления же синуса, косинуса занимает несколько микросекунд). Его можно заметно улучшить, если вычислить значения тригонометрических функций один раз и записать их в массив, а затем на каждом шаге обращаться к этому массиву. В данном случае уменьшается общее время вычислений за счет уменьшения числа подсчетов значений синуса и косинуса, а также соблюдается высокая точность вычислений. Однако он имеет небольшой недостаток: необходима дополнительная память для хранения вычисленных значений тригонометрических функций, однако количество требуемой памяти намного

меньше размера сигнала. Этот метод можно ускорить двумя способами. Во-первых, можно вычислить лишь значения синуса и косинуса, лежащие в первой четверти, а затем при помощи формул приведения вычислить остальные (получим ускорение около 4 раз). Во-вторых, можно вычислить лишь одно значение синуса и косинуса, а остальные значения можно получить последовательным домножением на первое (поворот текущего значения на минимальных угол). При таком подходе значительно уменьшается общее время вычисления экспонент (порядка N раз при N×N отсчетах сигнала), однако возникает незначительное снижение точности вычислений (ошибка в 8 знаке после запятой при 16777216 отсчетах). Для борьбы с ошибкой можно периодически явно вычислять значение синуса и косинуса, а затем осуществлять операцию поворота над ним.

Способ выделения памяти под двумерный массив

Самый простой в работе – это выделение памяти сначала под столбец указателей, а потом для каждого указателя под строку элементов. Такой метод дает удобную индексацию двумерного массива (строка, столбец), но строки в памяти могут находиться не последовательно, что может привести к частому считыванию данных из оперативной памяти в КЭШ. Однако есть более производительный способ – это выделение памяти под один большой одномерный массив. В нем строки данных будут расположены в памяти последовательно, что уменьшит вероятность не попадания в КЭШ и облегчит задачу распараллеливания. Но у этого способа есть недостаток – сложная индексация двумерного массива при написании программы (однако считается, что вычисление индекса происходит быстрее, чем не попадание в КЭШ в предыдущем случае). К тому же, большая длина одномерного массива может привести к выходу индекса за границу типа целых чисел (максимальное число: 2147483648), поэтому для индексации приходится использовать тип __int64 (максимальное число: 9223372036854775808). Другой способ решения этой

проблемы – это выделение одномерного массива как большого цельного куска памяти, выделение памяти под столбец указателей, и распределение указателей на строки по всему одномерному массиву подряд. Этот способ позволяет получить последовательное расположение строк в памяти и удобную двумерную индексацию (строка, столбец), однако несколько усложняет процесс выделения и освобождения памяти.

Пересылка данных между процессами

Первоначальное решение – использование стандартных пересылок типа «точка-точка» (MPI_Send, MPI_Resv). Они позволяют наглядно проследить передачу данными между процессами, но, так как основная модель распараллеливания - это наличие главного процесса, который пересылает и принимает данные от всех других, то происходит задержка при пересылке данных (процессы простаивают до тех пор, пока главный процесс не перешлет данные всем остальным). К тому же приходится вычислять тэг (уникальный номер – идентификатор) для каждого сообщения, чтобы данные не потерялись в процессе пересылки или вовремя достигли нужного процесса. Решение этой проблемы – использование групповых операций передачи данных (MPI_Bcast, MPI_Gather, MPI_Scatter). Данные операции оптимизированы под массовую рассылку данных произвольному число процессов (одинаковых данных всем, части данных каждому процессу, сбор частей данных от каждого процесса). Им не нужно определять тэг для каждого сообщения. Но основное требование к этим функциям – последовательное расположение данных в памяти, поэтому приходиться хранить данные в одномерном массиве и транспонировать его в двумерном БПФ по строчкам и столбцам. В двумерном БПФ по аналогу Кули-Тьюки приходится регистрировать новые типы данных (описывать местоположение начального элемента и сдвиг между элементами) для пересылки процессам строк с произвольным номером.

Увеличение числа процессов при распараллеливании

При этом увеличивается привлекаемая вычислительная мощность, однако возникают задержки при многочисленной пересылке данных по сети, возникает ситуация, когда скорость передачи данных по сети становится больше скорости их обработки, следовательно, при большом количестве процессов они будут простаивать. Но в случае запуска на одной локальной машине пересылка данных происходит не по сети, а в пределах оперативной памяти, что значительно ускоряет время работы программы.

Увеличение частоты пересылки данных

Алгоритм двумерного БПФ по строчкам и столбцам позволяет провести распараллеливание по данным (каждому процессу посылается строка или столбец данных, над которым он производит вычисления независимо от других процессов). Однако он требует двойного прохода массива (сначала по строчкам, потом по столбцам – или наоборот), что приводит к двойной пересылке и обработке исходного массива данных. Алгоритм двумерного БПФ по аналогу Кули-Тьюки позволяет провести распараллеливание по данным на каждом шаге рекурсии, однако с увеличение размера сигнала увеличивается число рекурсий, следовательно, и число общих пересылок. Поэтому алгоритм двумерного БПФ по аналогу Кули-Тьюки удобен при незначительном времени пересылки данных (например, запуск программы на нескольких ядрах одного процессора, при этом пересылка данных происходит в пределах оперативной памяти без участия сети), а алгоритм двумерного БПФ по строчкам и столбцам удобен при разделении узлов на несколько процессоров (за счет меньшего числа пересылок между процессами).

3.3 Результаты численного эксперимента в двумерном случае

Таблица 3 - Число отсчетов: Размер×Размер, один процессор - 8 ядер

Время в секундах

Размер	Число процессов	БПФ по строчкам и столбцам	БПФ по аналогу Кули-Тьюки	Ускорение Кули-Тьюки
1024	1	0,490	0,310	~1,6
	2	0,310	0,310	~1,0
	4	0,220	0,270	~0,8
	8	0,180	0,270	~0,7
	16	0,180	0,350	~0,5
2048	1	2,330	1,330	~1,7
	2	1,500	1,260	~1,2
	4	1,060	0,950	~1,1
	8	0,840	0,850	~0,9
	16	0,850	1,000	~0,8
4096	1	9,880	5,850	~1,7
	2	6,240	4,590	~1,4
	4	4,370	3,590	~1,2
	8	3,430	3,060	~1,1
	16	3,740	3,390	~1,1
8192	1	43,210	25,190	~1,7
	2	26,990	19,160	~1,4
	4	18,550	14,870	~1,2
	8	14,520	13,130	~1,1
	16	14,090	11,700	~1,2

Таблица 4 - Число отсчетов: Размер×Размер, несколько процессоров - 8 ядер

Время в секундах

Размер	Число процессов	БПФ по строчкам и столбцам		БПФ по аналогу Кули-Тьюки	
		2 процесса	4 процесса	2 процесса	4 процесса
1024	1	0,490	0,480	0,310	0,300
	2	0,610	0,600	1,170	1,150
	4	0,400	0,950	0,830	1,010
	8	0,290	0,330	0,630	0,840
	16	28,710	0,280	96,300	0,730
2048	1	2,330	2,310	1,340	1,320
	2	2,860	2,840	3,740	3,720
	4	2,080	2,420	2,800	3,610
	8	1,620	2,010	2,280	3,170
	16	154,330	1,750	-	71,360
4096	1	9,880	9,860	5,840	5,840
	2	9,730	9,710	13,020	12,970
	4	6,900	7,840	10,500	14,040
	8	5,400	6,480	36,800	120,610
	16	-	5,600	-	189,220
8192	1	43,170	42,950	25,300	26,530
	2	36,910	36,340	49,810	52,320
	4	25,940	29,070	41,810	88,310
	8	20,760	24,310	36,800	228,750
	16	-	35,770	-	-

3.4 Результаты численного эксперимента в трёхмерном случае

Таблица 5 - Сравнение времени работы алгоритмов трёхмерного БПФ при помощи комбинации одномерных и трёхмерное БПФ по аналогу Кули-Тьюки, в секундах

Размер	БПФ при помощи комбинации 1D БПФ	БПФ по аналогу Кули-Тьюки
32×32×32	0,006	0,002
64×64×64	0,045	0,019
128×128×128	0,298	0,134
256×256×256	2,850	1,225
512×512×512	29,840	10,658

ЗАКЛЮЧЕНИЕ

В ходе магистерской диссертации были закреплены знания, полученные в процессе обучения, приобретены новые знания в области параллельного программирования, разработаны параллельные алгоритмы двумерного быстрого преобразования Фурье, данные алгоритмы реализованы на языке программирования высокого уровня C++ с использованием библиотеки MPI, разработаны алгоритмы трёхмерного быстрого преобразования Фурье.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Алгоритмы быстрого преобразования Фурье FFT (fast Fourier transform). Принцип построения [Электронный ресурс]: 2008.
Режим доступа: www.dsplib.ru.

2) Антонов, А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие / Антонов, А.С. – Москва: Изд-во МГУ, 2004.-71 с.

3) Гонсалес, Р. Цифровая обработка изображений / Гонсалес Р., Вудс Р.: Москва: Техносфера, 2006 – 1072 с., 82-113 с.

4) Гришагин, В.А. Параллельное программирование на основе MPI. Учебное пособие / Гришагин В.А., Свистунов А.Н. – Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2005 -93 с.

5) Даджион, Д. Цифровая обработка многомерных сигналов: Пер. с англ. / Даджион Д., Мерсеро Р.: – Москва.: Мир, 1988. – 488 с., ил., 231-249.

6) Дубровина, О.В. Исследование алгоритмов быстрого преобразования Фурье двумерных сигналов для параллельных вычислительных систем [Электронный ресурс]: ДонНТУ, 2008.
Режим доступа: http://masters.donntu.edu.ua/2009/fvti/dubrovina/diss/index.htm.

7) Инструменты параллельного программирования в системах с общей памятью / Корняков К.В., Кустиков В.Д, Мееров И.Б., Сиднев А.А., Сысоев А.В., Шишков А.В.: – Нижегородский государственный университет им. Н.И.Лобачевского, 2010.

8) Комплекс Высокопроизводительных вычислений ИКИТ СФУ [Электронный ресурс]: Красноярск, 2011.
Режим доступа: www.cluster.sfu-kras.ru.

9) Кудрявцев, Л.Д. Краткий курс математического анализа. Т.2. Дифференциальное и интегральное исчисления функций многих переменных.

Гармонический анализ: учебник. / Кудрявцев Л.Д. – 3-е изд. перераб. – Москва: ФИЗМАТЛИТ, 2005. – 254 с., 274-296.

10) Малоземов, В.Н. Основы дискретного гармонического анализа Ч.2 / Малоземов В.Н., Машарский С.М. – Санкт-Петербург.: НИИММ, 2003.

11) Параллельный алгоритм дискретного преобразования Фурье [Электронный ресурс]: 2010.
Режим доступа: http://www.intuit.ru/department/supercomputing/ppmcp/12/.

12) Смоляная, Д. В. Исследование эффективной параллельной реализации быстрого преобразования Фурье для цифровых изображений [Электронный ресурс]: ДонНТУ, 2009.
Режим доступа: http://masters.donntu.edu.ua/2009/fvti/smolianaya/diss/index.htm.

13) Тутатчиков, В.С. Алгоритм параллельного быстрого преобразования Фурье для анализа изображений // XI Всероссийская конференция молодых ученых по математическому моделированию и информационным технологиям / Тутатчиков В.С., Носков М.В., Старовойтов А.В. – Новосибирск. 2010. 80 с.

14) Bailey, D.H. A High-Performance FFT Algorithm for Vector Supercomputers // International Journal of Supercomputer Applications / Bailey D.H., том.2, №.1, 1988, 82-87 с

15) Cooley, J.W. An algorithm for the machine calculation of complex Fourier series / Cooley J.W., and Tukey J. - Math. Comput. 19, 1965, C. 297-301.

16) CUDA CUFFT Library/ Официальный сайт разработчика [Электронный ресурс]:2010. Режим доступа: http://developer.download.nvidia.com/compute/cuda/21/toolkit/docs/CUFFT Library2.1.pdf.

17) Fast Fourier Transformation [Электронный ресурс]: Красноярск, 2011. - Режим доступа: http://www.fftw.org/.

18) Parallel Fourier Transformations using shared memory nodes // Solon Pissis. – 2008. – 78 с.

ПРИЛОЖЕНИЕ А

Текст программы двумерного БПФ

Файл Fourier.cpp - реализация функций быстрого преобразования Фурье по строчкам и столбцам и по аналогу Кули-Тьюки и некоторых вспомогательных к ним.

```
#ifndef FOURIER_CPP
#define FOURIER_CPP
#include <math.h>
#include <conio.h>
#include <iostream>
#include <mpi.h>
using namespace std;
const float M_PI = (float)3.14159265358979323846264338327 95;

//одномерное преобразование Фурье, N - размер массива, f - исходный вектор, в
который пишется преобразование, trig - вычисленные значения cos и sin, oper -
направление преобразования Фурье: -1 - прямое, +1 - обратное
void Fure_line_with_trig(int N, int s, float *Real, float *Image, float *Cos, float *Sin,
int oper)	//массив f - уже реверсирован
{

	int x1, x2;
	float Exp1_Real, Exp1_Image, Exp2_Real, Exp2_Image;//, temp_Real,
temp_Image;
	int v, Nv=N, dv=1, dv2, p, l;
	int index;
	for(v=1;v<=s;v++)
	{
		dv2=dv*2;
		Nv/=2;
		for(p=0; p<Nv; p++)
			for(l=0; l<dv; l++)
				{
					x1=l+p*dv2; x2=x1+dv;
					Exp1_Real = Real[x1];
					Exp1_Image = Image[x1];
					index = (N/(2*dv))*l;//l*Nv;
```

```
                                        Exp2_Real = Cos[index]*Real[x2] -
oper*Sin[index]*Image[x2];
                                        Exp2_Image = Cos[index]*Image[x2] +
oper*Sin[index]*Real[x2];
                                        Real[x1] = Exp1_Real + Exp2_Real;
                                        Image[x1] = Exp1_Image + Exp2_Image;
                                        Real[x2] = Exp1_Real - Exp2_Real;
                                        Image[x2] = Exp1_Image - Exp2_Image;
                                }
                dv *= 2;

        }
        float temp = sqrt((float)N);
                for(v=0;v<N;v++)
                {
                        Real[v] = Real[v] / temp;
                        Image[v] = Image[v] / temp;
                }
}

//БПФ по строчкам и столбцам
//для главного процесса передаются данные, для остальных Real = NULL, Image
= NULL
void Fure2(int Height, int log2Height, int Width, int log2Width, float **Real, float
**Image, bool flag, int rank, int size)
{
        int i, j, oper;
        float *Real_Array_Height, *Image_Array_Height, *Real_Array_Width,
*Image_Array_Width;
        if (flag)
                oper = 1;
        else
                oper = -1;
        Real_Array_Height = new float[Height];
        Image_Array_Height = new float[Height];
        Real_Array_Width = new float[Width];
        Image_Array_Width = new float[Width];
        float *Cos_Height, *Sin_Height, *Cos_Width, *Sin_Width, arg, TempSin,
TempCos;//, temp;
        Cos_Height = new float[(Height+1)/2];       //проверка при длине массива в 1
        Sin_Height = new float [(Height+1)/2];      // [0,Pi]
        arg = 2*M_PI/(float)Height;
        TempSin = sin(arg);
        TempCos = cos(arg);
```

```
Cos_Height[0] = 1;                    //cos(0)=1
Sin_Height[0] = 0;              //sin(0) = 0
Cos_Height[1] = TempCos;
Sin_Height[1] = TempSin;
for(i=2;i<Height/2;i++)
{
        Cos_Height[i] = Cos_Height[i-1]*TempCos - Sin_Height[i-1]*TempSin;
        Sin_Height[i] = Sin_Height[i-1]*TempCos + Cos_Height[i-1]*TempSin;
}
Cos_Width = new float [(Width+1)/2];      //проверка при длине массива в 1
Sin_Width = new float [(Width+1)/2];      // [0,Pi]
arg = 2*M_PI/(float)Width;
TempSin = sin(arg);
TempCos = cos(arg);
Cos_Width[0] = 1;               //cos(0)=1
Sin_Width[0] = 0;               //sin(0) = 0

Cos_Width[1] = TempCos;
Sin_Width[1] = TempSin;
for(i=2;i<Width/2;i++)
{
        Cos_Width[i] = Cos_Width[i-1]*TempCos - Sin_Width[i-1]*TempSin;
        Sin_Width[i] = Sin_Width[i-1]*TempCos + Cos_Width[i-1]*TempSin;
}

for(i=0;i<Height;i++)     //вызываем одномерное преобразование Фурье для
каждой строки
{
        for(j=0;j<Width;j++)      //заполняем массив a с транспонированием
        {
                Real_Array_Width[j] =
Real[i][j];//[Index_Height[i]][Index_Width[j]];
                Image_Array_Width[j] =
Image[i][j];//[Index_Height[i]][Index_Width[j]];;
        }


Fure_line_with_trig(Width,log2Width,Real_Array_Width,Image_Array_Width
,Cos_Width,Sin_Width,oper);
        for(j=0;j<Width;j++)      //считываем данные
        {
                Real[i][j] = Real_Array_Width[j];
                Image[i][j] = Image_Array_Width[j];
        }
```

```
	}
	for(i=0;i<Width;i++)	//вызываем одномерное преобразование Фурье для
каждой строки
	{
		for(j=0;j<Height;j++)
		{
			Real_Array_Height[j] =
Real[j][i];//[Index_Width[j]][Index_Height[i]];
			Image_Array_Height[j] =
Image[j][i];//[Index_Width[j]][Index_Height[i]];
		}
	Fure_line_with_trig(Height,log2Height,Real_Array_Height,Image_Array_Hei
ght,Cos_Height,Sin_Height,oper);
		for(j=0;j<Height;j++)
		{
			Real[j][i] = Real_Array_Height[j];
			Image[j][i] = Image_Array_Height[j];
		}
	}
	delete[] Cos_Height;
	delete[] Sin_Height;
	delete[] Cos_Width;
	delete[] Sin_Width;
	delete[] Real_Array_Height;
	delete[] Image_Array_Height;
	delete[] Real_Array_Width;
	delete[] Image_Array_Width;
}


//модифицированный алгоритм Кули-Тьюки
void Fourier_Kyli_Tuki (int N, int s, float *Real, float *Image, bool flag, int rank, int
size)
{
	int oper=0, x1, x2, y1, y2;
	int i, j;
	if(flag)
		oper = 1;
	else
		oper = -1;

	float Exe1_Real, Exe1_Image, Exe2_Real, Exe2_Image, Exe0_Real,
Exe0_Image, Exe3_Real, Exe3_Image, X1_Real, X1_Image, X2_Real, X2_Image;
	int v, Nv, dv, dv2, p1, l1, p2, l2;
```

```
int *Counts, *DisplsX1, *DisplsX2;   //число элементов и смещение
int *NumberLinesX, LinesPerProcess, *ArrayL1;
LinesPerProcess = (N/2)/size;
Counts = new int [size];
for(i=0;i<size;i++)
        Counts[i] = N;
DisplsX1 = new int[size];
DisplsX2 = new int[size];
NumberLinesX = new int[N];
ArrayL1 = new int[N];
float *LineX1Real, *LineX1Image, *LineX2Real, *LineX2Image;
LineX1Real = new float[N];
LineX1Image = new float[N];
LineX2Real = new float[N];
LineX2Image = new float[N];
int index1,index2;
float *Cos, *Sin, arg, CosTemp, SinTemp;
Cos = new float [(N+1)/2];      //проверка при длине массива в 1
Sin = new float [(N+1)/2];      // [0,Pi]
arg = 2*M_PI/(float)N;
float TempSin, TempCos;
TempSin = sin(arg);
TempCos = cos(arg);
Cos[0] = 1;                     //cos(0)=1
Sin[0] = 0;                     //sin(0) = 0
if(N!=2)
{
        Cos[1] = TempCos;
        Sin[1] = TempSin;
}
for(i=2;i<N/2;i++)
{
        Cos[i] = Cos[i-1]*TempCos - Sin[i-1]*TempSin;
        Sin[i] = Sin[i-1]*TempCos + Cos[i-1]*TempSin;
}
if(N>=4)
{
        Cos[N/4] = 0;
        Sin[N/4] = 1;
}
dv = 1;
dv2 = dv*2;
Nv = N/dv2;
for(v=1; v<=s; v++)
```

```
	{
		for(p1=0,i=0; p1<Nv; p1++)
			for(l1=0; l1<dv; l1++,i+=2)	//x - Height
			{
				//запоминаем нумерацию строк на текущей
итерации
				NumberLinesX[i] = l1+p1*dv2;
				NumberLinesX[i+1] = NumberLinesX[i]+dv;
				ArrayL1[i] = l1;
			}
			for(i=0;i<2*LinesPerProcess;i+=2)
			{
				//считаем смещение для пересылки
				for(j=0;j<size;j++)
				{
					DisplsX1[j] = NumberLinesX[i]*N;
					DisplsX2[j] = NumberLinesX[i+1]*N;
				}
					//делаем пересылку пар строк
				MPI_Barrier(MPI_COMM_WORLD);
				MPI_Scatterv(Real,
Counts,DisplsX1,MPI_FLOAT,LineX1Real,
N,MPI_FLOAT,0,MPI_COMM_WORLD);
	MPI_Scatterv(Image,Counts,DisplsX1,MPI_FLOAT,LineX1Image,N,MPI_FL
OAT,0,MPI_COMM_WORLD);
				MPI_Scatterv(Real,
Counts,DisplsX2,MPI_FLOAT,LineX2Real
,N,MPI_FLOAT,0,MPI_COMM_WORLD);
	MPI_Scatterv(Image,Counts,DisplsX2,MPI_FLOAT,LineX2Image,N,MPI_FL
OAT,0,MPI_COMM_WORLD);
				for(p2=0; p2<Nv; p2++)
					for(l2=0; l2<dv; l2++)	//y - Width
					{
						y1=l2+p2*dv2; y2=y1+dv;
						Exe0_Real = LineX1Real[y1];
						Exe0_Image = LineX1Image[y1];
						index1 = (N/(2*dv))*ArrayL1[i];
						Exe1_Real = Cos[index1]*LineX2Real[y1] -
oper*Sin[index1]*LineX2Image[y1];
						Exe1_Image = Cos[index1]*LineX2Image[y1]
+ oper*Sin[index1]*LineX2Real[y1];
						index2 = (N/(2*dv))*l2
						Exe2_Real = Cos[index2]*LineX1Real[y2] -
oper*Sin[index2]*LineX1Image[y2];
```

```
                                        Exe2_Image = Cos[index2]*LineX1Image[y2]
+ oper*Sin[index2]*LineX1Real[y2];
                                        CosTemp = Cos[index1]*Cos[index2] -
Sin[index1]*Sin[index2];
                                        SinTemp = oper*(Sin[index1]*Cos[index2] +
Cos[index1]*Sin[index2]);
                                        Exe3_Real = CosTemp*LineX2Real[y2] -
SinTemp*LineX2Image[y2];
                                        Exe3_Image = CosTemp*LineX2Image[y2] +
SinTemp*LineX2Real[y2];
                                        X1_Real = Exe0_Real + Exe1_Real;
                                        X1_Image = Exe0_Image + Exe1_Image;
                                        X2_Real = Exe2_Real + Exe3_Real;
                                        X2_Image = Exe2_Image + Exe3_Image;
                                        LineX1Real[y1] = X1_Real + X2_Real;
                                        LineX1Image[y1] = X1_Image + X2_Image;
                                        LineX1Real[y2] = X1_Real - X2_Real;
                                        LineX1Image[y2] = X1_Image - X2_Image;
                                        X1_Real = Exe0_Real - Exe1_Real;
                                        X1_Image = Exe0_Image - Exe1_Image;
                                        X2_Real = Exe2_Real - Exe3_Real;
                                        X2_Image = Exe2_Image - Exe3_Image;
                                        LineX2Real[y1] = X1_Real + X2_Real;
                                        LineX2Image[y1] = X1_Image + X2_Image;
                                        LineX2Real[y2] = X1_Real - X2_Real;
                                        LineX2Image[y2] = X1_Image - X2_Image;
                                    }
                                    //отсылаем обратно
                                MPI_Barrier(MPI_COMM_WORLD);
                                MPI_Gatherv(LineX1Real, N,MPI_FLOAT,Real,
Counts,DisplsX1,MPI_FLOAT,0,MPI_COMM_WORLD);
        MPI_Gatherv(LineX1Image,N,MPI_FLOAT,Image,Counts,DisplsX1,MPI_FL
OAT,0,MPI_COMM_WORLD);
                                MPI_Gatherv(LineX2Real, N,MPI_FLOAT,Real,
Counts,DisplsX2,MPI_FLOAT,0,MPI_COMM_WORLD);
        MPI_Gatherv(LineX2Image,N,MPI_FLOAT,Image,Counts,DisplsX2,MPI_FL
OAT,0,MPI_COMM_WORLD);
                        }
                dv *= 2;
                dv2 *= 2;
                Nv /= 2;
        }
        float temp = (float)N;
        for(i=0; i<N; i++)
```

```
		for(j=0; j<N; j++)
		{
			Real[i*N+j] /= temp;
			Image[i*N+j] /= temp;
		}
	delete[] Cos;
	delete[] Sin;
}

//БПФ по строчкам и столбцам, строчки и столбцы уже переставлены
//для главного процесса передаются данные, для остальных Real = NULL, Image
= NULL
void Fure2InLine(int Height, int log2Height, int Width, int log2Width, float
*RealLine, float *ImageLine, bool flag, int rank, int size)
{
	int i, j, oper, HeightPerProcess, WidthPerProcess;
	int PositionReal, PositionImage, SizeOfColumn, SizeOfColumnAsLine, tag;
	float *Real_Array_Height, *Image_Array_Height, *Real_Array_Width,
*Image_Array_Width,
		*Real_Array_Height_Packed, *Image_Array_Height_Packed, *ptrReal
= NULL, *ptrImage = NULL;
	if (flag)
		oper = 1;
	else
		oper = -1;
	HeightPerProcess = Height/size;		//целое число - проверяется в главной
функции
	WidthPerProcess = Width/size;
	Real_Array_Width = new float[Width];
	Image_Array_Width = new float[Width];
	float *Cos_Height, *Sin_Height, *Cos_Width, *Sin_Width, arg, TempSin,
TempCos;//, temp;
	Cos_Height = new float[(Height+1)/2];		//проверка при длине массива в 1
	Sin_Height = new float [(Height+1)/2];		// [0,Pi]
	arg = 2*M_PI/(float)Height;
	TempSin = sin(arg);
	TempCos = cos(arg);
	Cos_Height[0] = 1;				//cos(0)=1
	Sin_Height[0] = 0;			//sin(0) = 0
	if(Height != 2)
	{
		Cos_Height[1] = TempCos;
		Sin_Height[1] = TempSin;
	}
```

```
	for(i=2;i<Height/2;i++)
	{
		Cos_Height[i] = Cos_Height[i-1]*TempCos - Sin_Height[i-1]*TempSin;
		Sin_Height[i] = Sin_Height[i-1]*TempCos + Cos_Height[i-1]*TempSin;
	}
	Cos_Width = new float [(Width+1)/2];	//проверка при длине массива в 1
	Sin_Width = new float [(Width+1)/2];	// [0,Pi]
	arg = 2*M_PI/(float)Width;
	TempSin = sin(arg);
	TempCos = cos(arg);
	Cos_Width[0] = 1;		//cos(0)=1
	Sin_Width[0] = 0;		//sin(0) = 0
	if(Width!=2)
	{
		Cos_Width[1] = TempCos;
		Sin_Width[1] = TempSin;
	}
	for(i=2;i<Width/2;i++)	//считаем текущую комплексную экспоненту
поворотом предыдущей на угол
	{
		Cos_Width[i] = Cos_Width[i-1]*TempCos - Sin_Width[i-1]*TempSin;
		Sin_Width[i] = Sin_Width[i-1]*TempCos + Cos_Width[i-1]*TempSin;
	}

	ptrReal = RealLine;	//ставим указатель для пересылки на начало
данных
	ptrImage = ImageLine;

	for(i=0;i<HeightPerProcess;i++)	//по строчкам
	{
		MPI_Barrier(MPI_COMM_WORLD);
			//отправляем по строчке данных
		MPI_Scatter(ptrReal, Width,MPI_FLOAT,Real_Array_Width,
Width,MPI_FLOAT,0,MPI_COMM_WORLD);
	MPI_Scatter(ptrImage,Width,MPI_FLOAT,Image_Array_Width,Width,MPI_
FLOAT,0,MPI_COMM_WORLD);
			//считам БПФ над каждой строчкой
	Fure_line_with_trig(Width,log2Width,Real_Array_Width,Image_Array_Width
,Cos_Width,Sin_Width,oper);	//считам БПФ
		MPI_Barrier(MPI_COMM_WORLD);
			//принимаем по строчке данных
		MPI_Gather(Real_Array_Width, Width,MPI_FLOAT,ptrReal,
Width,MPI_FLOAT,0,MPI_COMM_WORLD);
```

```
	MPI_Gather(Image_Array_Width,Width,MPI_FLOAT,ptrImage,Width,MPI_F
LOAT,0,MPI_COMM_WORLD);
		if(rank == 0)
		{
			ptrReal+=Width;	//сдвигаем указатель на следующую строку
			ptrImage+=Width;
		}
	}
		//транспонируем данные
	int temp;
	if(rank == 0)
		for(i=0;i<Height;i++)
			for(j=i+1;j<Width;j++)
			{
				temp = RealLine[i*Height+j];
				RealLine[i*Height+j] = RealLine[j*Width+i];
				RealLine[j*Width+i] = temp;
				temp = ImageLine[i*Height+j];
				ImageLine[i*Height+j] = ImageLine[j*Width+i];
				ImageLine[j*Width+i] = temp;
			}
		//вызываем преобразования столбцов, как для строк
	ptrReal = RealLine;		//ставим указатель для пересылки на начало
данных
	ptrImage = ImageLine;
	for(i=0;i<HeightPerProcess;i++)		//по строчкам (столбцам)
	{
		MPI_Barrier(MPI_COMM_WORLD);
			//отправляем по строчке данных
		MPI_Scatter(ptrReal, Width,MPI_FLOAT,Real_Array_Width,
Width,MPI_FLOAT,0,MPI_COMM_WORLD);
	MPI_Scatter(ptrImage,Width,MPI_FLOAT,Image_Array_Width,Width,MPI_
FLOAT,0,MPI_COMM_WORLD);
			//считам БПФ над каждой строчкой

	Fure_line_with_trig(Width,log2Width,Real_Array_Width,Image_Array_Width
,Cos_Width,Sin_Width,oper);	//считам БПФ
		MPI_Barrier(MPI_COMM_WORLD);
			//принимаем по строчке данных
		MPI_Gather(Real_Array_Width, Width,MPI_FLOAT,ptrReal,
Width,MPI_FLOAT,0,MPI_COMM_WORLD);
	MPI_Gather(Image_Array_Width,Width,MPI_FLOAT,ptrImage,Width,MPI_F
LOAT,0,MPI_COMM_WORLD);
		if(rank == 0)
```

```
            {
                ptrReal+=Width;   //сдвигаем указатель на следующую строку
                ptrImage+=Width;
            }
        }
        delete[] Cos_Height;
        delete[] Sin_Height;
        delete[] Cos_Width;
        delete[] Sin_Width;
        delete[] Real_Array_Width;
        delete[] Image_Array_Width;
}
#endif //FOURIER_CPP
```

Файл main.cpp – ввод и сохранение данных, замер времени, вызов преобразования Фурье.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include <iostream>
#include <mpi.h>
using namespace std;
const double Pi = 3.14159265358979323846264338327 95;
//БПФ по строчкам и столбцам
void Fure2(int Height, int log2Height, int Width, int log2Width, float **Real, float
**Image, bool flag, int rank, int size);
//БПФ по строчкам и столбцам
void Fure2InLine(int Height, int log2Height, int Width, int log2Width, float
*RealLine, float *ImageLine, bool flag, int rank, int size);
//модифицированный алгоритм Кули-Тьюки
void Fourier_Kyli_Tuki (int N, int s, float *Real, float *Image, bool flag, int rank, int
size);

inline int IsLog2(int N)   //return -1 if N is not degree of 2
{
	int i=1,s=0;
	for(i=1;i<N;i*=2)
		s++;
	if(i == N)
		return s;
	else
		return -1;
}

int revs(int i, int s)
{
	int k;
	if(i==0 && s==0)
	return  0;
	else
	{
	if(int(i/2)==i/2.0)
		k=revs(int(i/2), s-1);
	else
		k=revs(int(i/2), s-1)+pow(2.0, s-1);
```

```
        return k;
        }
}

bool Save(char *FileName, int Height, int Width, float **Real, float **Image)
{
        FILE *out;
        out = fopen(FileName,"w");
        if(out == NULL)
        {
                cout << "Error! File " << FileName << " cann't be open for write!" <<
endl;
                return false;
        }
        int i,j;
        for(i=0;i<Height;i++)
        {
                for(j=0;j<Width;j++)
                        if(Image[i][j] == 0)
                                fprintf(out,"%.2f ",Real[i][j]);
                        else
                                if(Image[i][j] < 0)
                                        fprintf(out,"%.2f%.2fi ",Real[i][j],Image[i][j]);
                                else
                                        fprintf(out,"%.2f+%.2fi ",Real[i][j],Image[i][j]);
                fprintf(out,"\n");
        }
        fclose(out);
        return true;
}

bool Save(char *FileName, int Height, int Width, float *Real, float *Image)
{
        FILE *out;
        out = fopen(FileName,"w");
        if(out == NULL)
        {
                cout << "Error! File " << FileName << " cann't be open for write!" <<
endl;
                return false;
        }
        int i,j,k;
        for(i=0,k=0;i<Height;i++)
        {
```

```
		for(j=0;j<Width;j++,k++)
			if(Image[k] == 0)
				fprintf(out,"%.2f ",Real[k]);
			else
				if(Image[k] < 0)
					fprintf(out,"%.2f%.2fi ",Real[k],Image[k]);
				else
					fprintf(out,"%.2f+%.2fi ",Real[k],Image[k]);
		fprintf(out,"\n");
	}
	fclose(out);
	return true;
}

int main(int argc, char* argv[])
{
	int size,rank;
	MPI_Init(&argc,&argv);
	MPI_Comm_size(MPI_COMM_WORLD, &size); // number of processes
involved in run
	MPI_Comm_rank(MPI_COMM_WORLD, &rank); // my process id: 0 <=
rank < size
	if(IsLog2(size)<0)
	{
		cout << "Error! " << size << " is not degree of 2!" << endl;
		MPI_Finalize();
		return -1;
	}
	if(rank == 0)
		cout << "Number of process is " << size << endl;

	FILE *in,*out,*f;
	int Height = 0, Width = 0,
		tempHeight, tempWidth,
		Log2Height = 0, Log2Width = 0,
		i, j;
	_int64 k;
	char FileNameInput[20] = "input.txt",
		FileNameParam[20] = "param.txt",
		FileNameOutput[20] = "output.txt",
		FileNameKT[20] = "KT.txt",
		FileNameLC[20] = "LC.txt";
	float **Real = NULL, **Image = NULL, **RealData = NULL,
		*RealLine = NULL, *ImageLine = NULL;
```

```
	int Choise;
	clock_t
begin,end,WorkTime,CreateMemoryTime,LoadDataTime,SaveDataTime,Generation
DataTime,
		ReversionTime, FFT2KTTime,
FFT2LCTime,SaveFFT2KT,SaveFFT2LC;
	float second = 1.0 ;	//for calc second, 1000 for Win, 1000000 for Linux
	int RANGE_MIN = 0, RANGE_MAX = 255;
	int *IndexHeight=NULL,*IndexWidth=NULL;
	if(rank == 0)
	{
		cout << "1 - Create data2" << endl;
		cout << "2 - Load data2" << endl;
		cin >> Choise;

		switch(Choise)
		{
		case 1:
			//Create data2
			cout << "Input Height: ";
			cin >> Height;
			Log2Height = IsLog2(Height);
			if(Log2Height < 0)
			{
				cout << "Error in height!" << endl;
				return -1;
			}
			cout << "Input Width: ";
			cin >> Width;
			Log2Width= IsLog2(Width);
			if(Log2Width < 0)
			{
				cout << "Error in Width!" << endl;
				return -1;
			}
			f = fopen(FileNameParam,"w");
			//f = fopen("param.txt","w");
			if(f == NULL)
			{
				cout << "Error in open file " << FileNameParam << " for
write parametrs" << endl;
				return -1;
			}
			fprintf(f,"%d %d\n",Height,Width);
```

```
fclose(f);
begin = clock();
RealData = new float*[Height];
Real = new float*[Height];
Image = new float*[Height];
//FourierReal = new float*[Height];
//FourierImage = new float*[Height];
for(i=0;i<Height;i++)
{
	RealData[i] = new float[Width];
	Real[i] = new float[Width];
	Image[i] = new float[Width];
	//FourierReal[i] = new float[Width];
	//FourierImage[i] = new float[Width];
	for(j=0;j<Width;j++)
	{
		Real[i][j] = 0;
		Image[i][j] = 0;
		//FourierReal[i][j] = 0;
		//FourierImage[i][j] = 0;
	}
}
RealLine = new float[Height*Width];
ImageLine = new float[Height*Width];
for(k=0;k<Height*Width;k++)
{
	RealLine[k] = 0;
	ImageLine[k] = 0;
}
end = clock();
CreateMemoryTime = (end - begin)/second;
cout << "CreateMemoryTime = " << CreateMemoryTime << endl;
	begin = clock();
srand( (unsigned)time( NULL ) );
for(i=0;i<Height;i++)
	for(j=0;j<Width;j++)
		RealData[i][j] = (((float) rand() / (float)
RAND_MAX) * RANGE_MAX + RANGE_MIN);
end = clock();
GenerationDataTime = (end - begin)/second;
cout << "GenerationDataTime = " << GenerationDataTime <<
endl;
in = fopen(FileNameInput,"w");
if(in == NULL)
```

```
{
	cout << "Error! File " << FileNameInput << " cann't be
open for write data!" << endl;
	return -1;
}
begin = clock();
for(i=0;i<Height;i++)
{
	for(j=0;j<Width;j++)
		fprintf(in,"%d ",(int)RealData[i][j]);
	fprintf(in,"\n");
}
end = clock();
fclose(in);
SaveDataTime = (end - begin)/second;
cout << "SaveDataTime = " << SaveDataTime << endl;
break;
case 2:
	//Load data2
	f = fopen(FileNameParam,"r");
	if(f == NULL)
	{
		cout << "Error! File " << FileNameParam<< " cann't be
open for read data!" << endl;
			//need delete memory in ideal case
		return -1;
	}
	fscanf(f,"%d%d",&Height,&Width);
	fclose(f);
	Log2Height = IsLog2(Height);
	Log2Width = IsLog2(Width);
	if(Log2Height<0 && Log2Width<0)
	{
		cout << "Error! Incorrect parametrs!" << endl;
		MPI_Finalize();
		return -1;
	}
	begin = clock();
	RealData = new float*[Height];
	Real = new float*[Height];
	Image = new float*[Height];
	for(i=0;i<Height;i++)
	{
		RealData[i] = new float[Width];
```

```
                Real[i] = new float[Width];
                Image[i] = new float[Width];
                for(j=0;j<Width;j++)
                {
                        Real[i][j] = 0;
                        Image[i][j] = 0;
                }
        }
        RealLine = new float[Height*Width];
        ImageLine = new float[Height*Width];
        for(k=0;k<Height*Width;k++)
        {
                RealLine[k] = 0;
                ImageLine[k] = 0;
        }
        end = clock();
        CreateMemoryTime = (end - begin)/second;
        cout << "CreateMemoryTime = " << CreateMemoryTime << endl;

        in = fopen(FileNameInput,"r");
        if(in == NULL)
        {
                cout << "Error! File " << FileNameInput << " cann't be
open for read data!" << endl;
                        //need delete memory in ideal case
                return -1;
        }
        begin = clock();
        for(i=0;i<Height;i++)
                for(j=0;j<Width;j++)
                        fscanf(in,"%f",&RealData[i][j]);
        end = clock();
        LoadDataTime = (end - begin)/second;
        cout << "LoadDataTime = " << LoadDataTime << endl;
        fclose(in);
        break;
    default:
        cout <<"Error in Choise!\n" << endl;
        MPI_Finalize();
        return -1;
    }
    cout << "Height = " << Height << ", Width = " << Width << endl;
    IndexHeight = new int[Height];
    IndexWidth = new int[Width];
```

```
		for(i=0;i<Height;i++)
			IndexHeight[i] = revs(i,Log2Height);
		for(j=0;j<Width;j++)
			IndexWidth[j] = revs(j,Log2Width);
		begin = clock();
		for(i=0,k=0;i<Height;i++)
			for(j=0;j<Width;j++,k++)
			{
				Real[i][j] = RealData[IndexHeight[i]][IndexWidth[j]];
				RealLine[k] = RealData[IndexHeight[i]][IndexWidth[j]];
			}
		end = clock();
		ReversionTime = (end - begin)/second;
	}	//rank == 0
	MPI_Barrier(MPI_COMM_WORLD);
	MPI_Bcast(&Height,1,MPI_INT,0,MPI_COMM_WORLD);
	MPI_Barrier(MPI_COMM_WORLD);
	MPI_Bcast(&Width,1,MPI_INT,0,MPI_COMM_WORLD);
	MPI_Barrier(MPI_COMM_WORLD);
	MPI_Bcast(&Log2Height,1,MPI_INT,0,MPI_COMM_WORLD);
	MPI_Barrier(MPI_COMM_WORLD);
	MPI_Bcast(&Log2Width,1,MPI_INT,0,MPI_COMM_WORLD);
	if(rank == 0)
	{
		cout << "FFT2 Line and Columns" << endl;
		begin = clock();
	}
	Fure2InLine(Height,Log2Height,Width,Log2Width,RealLine,ImageLine,true,r
ank,size);
	if(rank == 0)
	{
		end = clock();
		FFT2LCTime = (end - begin)/second;
		cout << "FFT2LCTime = " << FFT2LCTime << endl;
	}
	if(rank == 0)
	{
		begin = clock();
		for(i=0,k=0;i<Height;i++)
			for(j=0;j<Width;j++,k++)
			{
				RealLine[k] = RealData[IndexHeight[i]][IndexWidth[j]];
				ImageLine[k] = 0;
			}
```

```
		end = clock();
		ReversionTime = (end - begin)/second;
	}
	if(rank == 0)
	{
		cout << "FFT2 Kyli-Tuki" << endl;
		begin = clock();
	}		Fourier_Kyli_Tuki(Height,Log2Height,RealLine,ImageLine,true,
rank, size);
	if(rank == 0)
	{
		end = clock();
		FFT2KTTime = (end - begin)/second;
		cout << "FFT2KTTime = " << FFT2KTTime << endl;
	}
	MPI_Barrier(MPI_COMM_WORLD);
	cout << rank << ": End of program" << endl;
	MPI_Finalize();
	return 1;
}
```

Файл param.txt – хранит два числа: размеры сигнала, на данный момент программа работает только с «квадратным» сигналом, сторона которого – степень 2.

4 4

Файл input.txt – содержит сам сигнал.

42 61 207 251

115 164 26 174

107 77 73 33

156 124 86 150

Файл KT.txt – содержит результат двумерного преобразования Фурье по аналогу Кули-Тьюки

588.51 -83.56-51.44i 24.99 -83.56+51.44i

-92.65+9.86i -73.75+46.17i 11.14+56.38i 66.46+2.84i

-100.01 8.19-59.84i -27.22 8.19+59.84i

-92.65-9.86i 66.46-2.84i 11.14-56.38i -73.75-46.17i

Файл LC.txt – содержит результат двумерного преобразования Фурье по строчкам и столбцам

588.51 -92.65+9.86i -100.01 -92.65-9.86i

-83.40-50.96i -73.59+46.66i 8.36-59.35i 66.63-2.35i

25.10 11.00+56.38i -27.10 11.00-56.38i

-83.19+50.95i 66.45+2.69i 8.19+59.05i -73.45-46.69i

ПРИЛОЖЕНИЕ Б

Текст программы трёхмерного БПФ

Файл FFT_1D.h – содержит описание класса для вычисления одномерного БПФ Кули-Тьюки.

```
class FFT_1D
{
private:
	int Length;
	//длина одномерного сигнала
	int Log2Length;
	//log2 Length
	double *Real;
	//Вещественная часть сигнала
	double *Image;
	//Мнимая часть сигнала
	double *Cos;
	//массив косинусов
	double *Sin;
	//массив синусов
	int *Revs;
	//массив реверсированных индексов

public:
	FFT_1D();
	//конструктор
	~FFT_1D();
	//деструктор

	bool Init(char *FileName);
	//инициализация - чтение размера сигнала из файла параметров
	bool Init(int length);
	//инициализация - чтение размера сигнала из параметра
	bool CreateMemory();
	//выделение памяти
	bool DeleteMemory();
	//удаление памяти
	bool Clear();
	//очистка данных
```

```
	bool Revers();
	//реверс индексов массива
	bool InputData(char *Param, char *FileName,
	bool GenerateData, bool rev);	//чтение
данных из файла с реверсированием
	bool ChangeData(char *Param, char *FileName);	//изменение
размера входного сигнала
	bool OutputData(char *FileName);	//запись
результатов в файл
	bool CalcExponent();
	//вычисление комплексной экспоненты
	bool DFT(bool direct);
	//прямое вычисление преобразования Фурье
	bool FFT(bool direct);
	//БПФ Кули-Тьюки
	bool FFTM(bool direct);
	//БПФ Кули-Тьюки модифицированный
	bool SetData(double *real, double *image);
	//устанавливает данные
	bool GetData(double *real, double *image);	//получает
данные
};
```

Файл FFT_3D.h – содержит описание класса для вычисления трёхмерного БПФ Кули-Тьюки.

```
class FFT_3D
{
private:
	int Length, Width, Height;
	//длина, ширина, высота трехмерного сигнала
	int Log2Length, Log2Width, Log2Height;	//log2
Length
	double ***Real;
	//Вещественная часть сигнала
	double ***Image;
	//Мнимая часть сигнала
	double *CosLength, *CosWidth, *CosHeight;	//массив
косинусов
	double *SinLength, *SinWidth, *SinHeight;	//массив
синусов
	int *RevsLength, *RevsWidth, *RevsHeight;	//массив
реверсированных индексов
```

```
public:
	FFT_3D();
	//конструктор
	~FFT_3D();
	//деструктор

	bool Init(char *FileName);
	//инициализация - чтение размера сигнала из файла параметров
	bool CreateMemory();
	//выделение памяти
	bool DeleteMemory();
	//удаление памяти
	bool Clear();
	//очистка данных
	bool ReversInit();
	//реверс индексов массива
	bool Revers();
	//реверс элементов массива
	bool InputData(char *Param, char *FileName,
	bool GenerateData, bool rev);	//чтение
данных из файла с реверсированием
	bool ChangeData(char *Param, char *FileName);	//изменение
размера входного сигнала
	bool OutputData(char *FileName);	//запись
результатов в файл
	bool CalcExponent();
	//вычисление комплексной экспоненты
	bool DFT(bool direct);
	//прямое вычисление преобразования Фурье
	bool FFT(bool direct);
	//БПФ Кули-Тьюки
	bool FFTper1D(bool direct);
	//БПФ при помощи комбинации одномерных БПФ
};
```

Файл FFT_1D.cpp – содержит реализацию методов класса FFT_1D вычисления одномерного БПФ.

```
#include <stdio.h>
#include <time.h>
#include <iostream>
#include <math.h>
#include "FFT_1D.h"

using namespace std;
```

```
int IsLog2(int N);  //main
int revs(int i, int s);         //main

//конструктор
FFT_1D::FFT_1D()
{
        Length = 0;
        Log2Length = -1;
        Real = NULL;
        Image = NULL;
        Cos = NULL;
        Sin = NULL;
}

//деструктор
FFT_1D::~FFT_1D()
{
        DeleteMemory();
        Length = 0;
        Log2Length = -1;
        Real = NULL;
        Image = NULL;
        Cos = NULL;
        Sin = NULL;
}

//инициализация - чтение размера сигнала из файла параметров
bool FFT_1D::Init(char *FileName)
{
        if(Length > 0 || Log2Length >= 0)      //память уже выделена
                return false;
        int N,s;
        FILE *f;
        if(FileName == NULL)
        {
                printf("Error! FileName is empty!\n");
                return false;
        }
        f = fopen(FileName,"r");
        if(f == NULL)
        {
                printf("Error! File %s can not be open!\n",FileName);
                return false;
```

```
	}
	fscanf(f,"%d",&N);
	if(N < 0)
		return false;
	s = IsLog2(N);
	if(s<0)
		return false;
	Length = N;
	Log2Length = s;
	return true;
}

//инициализация - чтение размера сигнала из параметра
bool FFT_1D::Init(int length)
{
	if (length<0)
		return false;
	int s = IsLog2(length);
	if(s<0)
		return false;
	Length = length;
	Log2Length = s;
	return true;
}

//выделение памяти под N = 2^S
bool FFT_1D::CreateMemory()
{
	if(Length < 1 || Log2Length<0)//не задан размер
		return false;
	//если память уже выделена, то удаляем её
	if(Real != NULL || Image != NULL)
		DeleteMemory();
	Real = new double[Length];
	Image = new double[Length];
	Cos = new double[Length];
	Sin = new double[Length];
	Revs = new int[Length];
	return true;
}

//удаление памяти
bool FFT_1D::DeleteMemory()
{
```

```
	//если память не выделена
	if (Length == 0)
		return true;
	delete[] Real;
	Real = NULL;
	delete[] Image;
	Image = NULL;
	delete[] Cos;
	Cos = NULL;
	delete[] Sin;
	Sin = NULL;
	delete[] Revs;
	Revs = NULL;
	Length = 0;
	Log2Length = -1;
	return true;
}

//очистка данных
bool FFT_1D::Clear()
{
	for(int i=0;i<Length;i++)
	{
		Real[i] = 0;
		Image[i] = 0;
	}
	return true;
}

//реверс индексов массива
bool FFT_1D::Revers()
{
	for(int i=0;i<Length;i++)
		Revs[i] = revs(i,Log2Length);
	return true;
}

//чтение данных из файла, если GenerateData = true, то создаем новые данные
рандомно и пишем их в файл с реверсированием, если rev = true
bool FFT_1D::InputData(char *Param, char *FileName, bool GenerateData, bool
rev)
{
	if(Param == NULL)
	{
```

```
		printf("Error! Param is empty!\n");
		return false;
	}
	if(FileName == NULL)
	{
		printf("Error! FileName is empty!\n");
		return false;
	}
	int RANGE_MIN = 0, RANGE_MAX = 255;
	FILE *f;
	f = fopen(Param,"r");
	if(f == NULL)
	{
		printf("Error! File %s can not be open!\n",FileName);
		return false;
	}
	int N = -1;
	fscanf(f,"%d",&N);
	fclose(f);
	if(N != Length)
	{
		printf("Incorrect size!\n");
		return false;
	}
	double temp;
	if(GenerateData == true)
	{
		f = fopen(FileName,"w");
		if(f == NULL)
		{
			printf("Error! File %s can not be open!\n",FileName);
			return false;
		}
		//создаем новые данные рандомно и пишем их в файл
		srand( (unsigned)time( NULL ) );
		for(int i=0;i<Length;i++)
		{
			if(rev == true)
			{
				temp = (((double) rand() / (double) RAND_MAX) *
RANGE_MAX + RANGE_MIN);
				Image[i] = 0;
				fprintf(f,"%.0f ",temp);
				//Real[Revs[i]]
```

```
                }
                else
                {
                        Real[i] = (((double) rand() / (double) RAND_MAX) *
RANGE_MAX + RANGE_MIN);
                        Image[i] = 0;
                        fprintf(f,"%.0f ",Real[i]);
                }
        }
        fprintf(f,"\n");
        fclose(f);
    }
    //else  //читаем из файла всегда
    {
        //читаем данные из файла
        f = fopen(FileName,"r");
        if(f == NULL)
        {
            printf("Error! File %s can not be open!\n",FileName);
            return false;
        }
        for(int i=0;i<Length;i++)
        {
            if(rev == true)
            {
                fscanf(f,"%lf", &temp);
                Real[Revs[i]] = temp;
                Image[i] = 0;
            }
            else
            {
                fscanf(f,"%lf", &temp);
                Real[i] = temp;
                Image[i] = 0;
            }
        }
        fclose(f);
    }
    return true;
}

//изменение размера входного сигнала
bool FFT_1D::ChangeData(char *Param, char *FileName)
{
```

```
if(FileName == NULL)
{
	printf("Error! Param is empty!\n");
	return false;
}
if(FileName == NULL)
{
	printf("Error! FileName is empty!\n");
	return false;
}

int N = -1, Log2N = -1;
while (true)
{
	printf("Input Length as degree 2: ");
	scanf("%d",&N);
	Log2N = IsLog2(N);
	if(Log2N < 0)
		printf("Incorrect N!\n");
	else
		break;
}
DeleteMemory();
Length = N;
Log2Length = Log2N;
CreateMemory();
FILE *f;
f = fopen(Param,"w");
if(f == NULL)
{
	printf("Error! File %s can not be open!\n",Param);
	return false;
}
fprintf(f,"%d %d\n",&Length,&Log2Length);
fclose(f);

int RANGE_MIN = 0, RANGE_MAX  = 255;
srand( (unsigned)time( NULL ) );
f = fopen(FileName,"w");
if(f == NULL)
{
	printf("Error! File %s can not be open!\n",FileName);
	return false;
}
```

```
	for(int i=0;i<Length;i++)
	{
		Real[i] = (((double) rand() / (double) RAND_MAX) * RANGE_MAX +
RANGE_MIN);
		Image[i] = 0;
		fprintf(f,"%.0f ",Real[i]);
	}
	fprintf(f,"\n");
	fclose(f);
	return true;
}

//запись результатов в файл
bool FFT_1D::OutputData(char *FileName)
{
	if(FileName == NULL)
	{
		printf("Error! FileName is empty!\n");
		return false;
	}
	FILE *f;
	f = fopen(FileName,"w");
	if(f == NULL)
	{
		printf("Error! File %s can not be open!\n",FileName);
		return false;
	}
	for(int i=0;i<Length;i++)
	{
		if(Image[i] == 0)
			fprintf(f,"%.2f ", Real[i]);
		else
			if(Image[i] < 0)
				fprintf(f,"%.2f%.2fi ", Real[i], Image[i]);
			else
				fprintf(f,"%.2f+%.2fi ", Real[i], Image[i]);
	}
	fprintf(f,"\n");
	fclose(f);
	return true;
}

//вычисление комплексной экспоненты
bool FFT_1D::CalcExponent()
```

```
{
	//заполняем массивы значениями 2*Pi*i/N, i=[0,N-1]
	if(Length < 1)
	{
		printf("Incorrect size!\n");
		return false;
	}

	if(Length == 1)
	{
		Cos[0] = 1.0;
		Sin[0] = 0;
		return true;
	}

	if(Length == 2)
	{
		Cos[0] = 1.0;
		Sin[0] = 0;
		Cos[1] = -1.0;
		Sin[1] = 0;
		return true;
	}
	if(Length == 4)
	{
		Cos[0] = 1.0;		//0
		Sin[0] = 0;
		Cos[1] = 0;		//Pi/2
		Sin[1] = 1.0;
		Cos[2] = -1.0;		//Pi
		Sin[2] = 0;
		Cos[3] = 0;		//3Pi/2
		Sin[3] = -1.0;
		return true;
	}
	const double M_PI = acos(-1.0);
	double TempCos = cos(2*M_PI/Length);	//значение на которое надо
повернуть комплексное число
	double TempSin = sin(2*M_PI/Length);
	for(int i=0;i<Length/4;i++)
	{
		if(i%32 == 0)	//корректировка точности: каждое 32 значение
вычисляем напрямую
		{
```

```
                Cos[i] = cos(2*M_PI*i/Length);
                Sin[i] = sin(2*M_PI*i/Length);
            }
            else                    //остальные поворотом предыдущего угла на
единичный
            {
                Cos[i] = Cos[i-1]*TempCos - Sin[i-1]*TempSin;
                Sin[i] = Sin[i-1]*TempCos + Cos[i-1]*TempSin;
            }
        }

        for(int i=0;i<Length/4;i++)
        {
            Cos[i + Length/4] = - Sin[i];
            Sin[i + Length/4] = Cos[i];
            Cos[i + Length/2] = - Cos[i];
            Sin[i + Length/2] = - Sin[i];
            Cos[i + 3*Length/4] = Sin[i];
            Sin[i + 3*Length/4] = - Cos[i];
        }
        return true;
}

//прямое вычисление преобразования Фурье
bool FFT_1D::DFT(bool direct)
{
        int arg = 0, TempLength, TempLog2Length;
        double *RealData = NULL, *ImageData = NULL, *TempReal = NULL,
*TempImage = NULL;
        //выделяем память под результат преобразования Фурье
        RealData = new double[Length];
        ImageData = new double[Length];
        //меняем местами массивы исходный и результирующий, чтобы результат
писался в данные класса
        TempReal = RealData;
        TempImage = ImageData;
        RealData = Real;
        ImageData = Image;
        Real = TempReal;
        Image = TempImage;
        //заполняем значение 0
        Clear();
        int oper;
        if(direct == true)
```

```
		oper = -1;
	else
		oper = 1;
	for(int i=0;i<Length;i++)
	{
		for(int j=0;j<Length;j++)
		{
			//подсчитываем показатель экспоненты
			arg = (i*j)%Length;
			//Real[i] += RealData[j]*Cos[arg] - oper * ImageData[j]*Sin[arg];
			Real[i] += Cos[arg]*RealData[j] - oper*Sin[arg]*ImageData[j];
			Image[i] += Cos[arg]*ImageData[j] + oper*Sin[arg]*RealData[j];
		}
	}

	delete[] RealData;
	delete[] ImageData;
	return true;
}

//БПФ Кули-Тьюки
bool FFT_1D::FFT(bool direct)
{
	int x1,x2, oper, arg;
	double Exp1_Real, Exp1_Image, Exp2_Real, Exp2_Image;
	int v, Nv = Length, dv = 1, dv2,p,l;
	if(direct == true)
		oper = -1;
	else
		oper = 1;
	for(v = 1; v <= Log2Length; v++)
	{
		dv2 = dv * 2;
		Nv /= 2;
		for(p = 0; p < Nv; p++)
			for(l = 0; l < dv; l++)
				{
					x1 = l + p * dv2;
					x2 = x1 + dv;
					Exp1_Real = Real[x1];
					Exp1_Image = Image[x1];
					arg = (Length / (2 * dv)) * l;
					Exp2_Real = Cos[arg] * Real[x2] - oper * Sin[arg] *
Image[x2];
```

```
                                Exp2_Image = Cos[arg] * Image[x2] + oper *
Sin[arg] * Real[x2];
                                Real[x1] = Exp1_Real + Exp2_Real;
                                Image[x1] = Exp1_Image + Exp2_Image;
                                Real[x2] = Exp1_Real - Exp2_Real;
                                Image[x2] = Exp1_Image - Exp2_Image;
                        }
                dv *= 2;
        }
        double temp = (double)Length;
        if(direct == false)
                for(v = 0; v < Length; v++)
                {
                        Real[v] = Real[v] / temp;
                        Image[v] = Image[v] / temp;
                }
        return true;
}

//БПФ Кули-Тьюки модифицированный
bool FFT_1D::FFTM(bool direct)
{
        int *Arg = NULL;
        int x1,x2, oper, arg;
        double Exp1_Real, Exp1_Image, Exp2_Real, Exp2_Image;
        int v, Nv = Length, dv = 1, dv2,p,l;
        if(direct == true)
                oper = -1;
        else
                oper = 1;
        Arg = new int[Length];
        for(int i=0;i<Length;i++)
                Arg[i] = 0;
        for(v = 1; v < Log2Length; v++)
        {
                dv2 = dv * 2;
                Nv /= 2;
                for(p = 0; p < Nv; p++)
                        for(l = 0; l < dv; l++)
                                {
                                        x1 = l + p * dv2;
                                        x2 = x1 + dv;
                                        Exp1_Real = Real[x1];
                                        Exp1_Image = Image[x1];
```

```
                                arg = (Length / (2 * dv)) * l;
                                //Exp1 = Real[x1]
                                //Exp2 = Real[x2]*exp(oper*arg)
                                //Exp2_Real = Cos[arg] * Real[x2] - oper * Sin[arg] *
Image[x2];
                                Exp2_Real = Real[x2];
                                //Exp2_Image = Cos[arg] * Image[x2] + oper *
Sin[arg] * Real[x2];
                                Exp2_Image = Image[x2];
                                Arg[x1] += 0;
                                Arg[x2] += oper*arg;
                                Real[x1] = Exp1_Real + Exp2_Real;
                                Image[x1] = Exp1_Image + Exp2_Image;
                                Real[x2] = Exp1_Real - Exp2_Real;
                                Image[x2] = Exp1_Image - Exp2_Image;
                        }
            dv *= 2;
        }

        double TempReal, TempImage;
        for(int i=0;i<Length;i++)
        {
            TempReal = Real[i];
            TempImage = Image[i];
            arg = Arg[i]%Length;
            if( arg < 0)
                arg += Length;
            Real[i] = Cos[arg]*TempReal - Sin[arg]*TempImage;
            Image[i] = Cos[arg]*TempImage + Sin[arg]*TempReal;
        }

        dv2 = dv * 2;
        Nv /= 2;
        for(p = 0; p < Nv; p++)
            for(l = 0; l < dv; l++)
                {
                    x1 = l + p * dv2;
                    x2 = x1 + dv;
                    Exp1_Real = Real[x1];
                    Exp1_Image = Image[x1];
                    arg = (Length / (2 * dv)) * l;
                    //Exp1 = Real[x1]
                    //Exp2 = Real[x2]*exp(oper*arg)
```

```
                                Exp2_Real = Cos[arg] * Real[x2] - oper * Sin[arg] *
Image[x2];
                                //Exp2_Real = Real[x2];
                                Exp2_Image = Cos[arg] * Image[x2] + oper * Sin[arg] *
Real[x2];
                                //Exp2_Image = Image[x2];
                                Arg[x1] += 0;
                                Arg[x2] += oper*arg;
                                Real[x1] = Exp1_Real + Exp2_Real;
                                Image[x1] = Exp1_Image + Exp2_Image;
                                Real[x2] = Exp1_Real - Exp2_Real;
                                Image[x2] = Exp1_Image - Exp2_Image;
                        }
        dv *= 2;


        //double temp = sqrt((double)Length);
        //for(v = 0; v < Length; v++)
        //{
        //      Real[v] = Real[v] / temp;
        //      Image[v] = Image[v] / temp;
        //}
        return true;
}

//устанавливает данные
bool FFT_1D::SetData(double *real, double *image)
{
        Real = real;
        Image = image;

        return true;
}

//получает данные
bool FFT_1D::GetData(double *real, double *image)
{
        real = Real;
        image = Image;

        return true;
}
```

Файл FFT_3D.cpp – содержит реализацию методов класса FFT_3D для вычисления трёхмерного БПФ, в том числе реализацию метода для вычисления трёхмерного БПФ по аналогу Кули-Тьюки.

```
#include <stdio.h>
#include <time.h>
#include <iostream>
#include <math.h>
#include "FFT_3D.h"
#include "FFT_1D.h"

using namespace std;

int IsLog2(int N);  //main
int revs(int i, int s);        //main

//конструктор
FFT_3D::FFT_3D()
{
        Length = 0;
        Log2Length = -1;
        Real = NULL;
        Image = NULL;
        CosLength = NULL;
        SinLength = NULL;
        CosWidth = NULL;
        SinWidth = NULL;
        CosHeight = NULL;
        SinHeight = NULL;
}

//деструктор
FFT_3D::~FFT_3D()
{
        DeleteMemory();
        Length = 0;
        Log2Length = -1;
        Real = NULL;
        Image = NULL;
        CosLength = NULL;
        SinLength = NULL;
        CosWidth = NULL;
        SinWidth = NULL;
        CosHeight = NULL;
```

```
	SinHeight = NULL;
}

//инициализация - чтение размера сигнала из файла параметров
bool FFT_3D::Init(char *FileName)
{
	if(Length > 0 && Log2Length > 0 && Width > 0 && Log2Width > 0 &&
Height > 0 && Log2Height > 0)	//память уже выделена
		return false;
	int length, log2length, width, log2width, heigth, log2height;
	FILE *f;
	if(FileName == NULL)
	{
		printf("Error! FileName is empty!\n");
		return false;
	}
	f = fopen(FileName,"r");
	if(f == NULL)
	{
		printf("Error! File %s can not be open!\n",FileName);
		return false;
	}
	fscanf(f,"%d",&length);
	if(length < 0)
		return false;
	log2length = IsLog2(length);
	if(log2length<0)
		return false;
	Length = length;
	Log2Length = log2length;

	fscanf(f,"%d",&width);
	if(width < 0)
		return false;
	log2width = IsLog2(width);
	if(log2width<0)
		return false;
	Width = width;
	Log2Width = log2width;

	fscanf(f,"%d",&heigth);
	if(heigth < 0)
		return false;
	log2height = IsLog2(heigth);
```

```
	if(log2height<0)
		return false;
	Height = heigth;
	Log2Height = log2height;

	return true;
}

//выделение памяти под N = 2^S
bool FFT_3D::CreateMemory()
{
	if(Length < 0 && Log2Length < 0 && Width < 0 && Log2Width < 0 &&
Height < 0 && Log2Height < 0)	//не задан размер
		return false;
	//если память уже выделена, то удаляем её
	if(Real != NULL || Image != NULL)
		DeleteMemory();
	double *ContiniousArrayReal, *ContiniousArrayImage;
	ContiniousArrayReal = new double [Length*Width*Height];
	ContiniousArrayImage = new double [Length*Width*Height];
	Real = new double**[Length];
	Image = new double**[Length];
	for(int i=0;i<Length;i++)
	{
		Real[i] = new double*[Width];
		Image[i] = new double*[Width];
		for(int j=0;j<Width;j++)
		{
			Real[i][j] = &ContiniousArrayReal[i*Width*Length + j*Width];
			Image[i][j] = &ContiniousArrayImage[i*Width*Length +
j*Width];
		}
	}
	ContiniousArrayReal = NULL;
	ContiniousArrayImage = NULL;
	CosLength = new double[Length];
	SinLength = new double[Length];
	CosWidth = new double[Width];
	SinWidth = new double[Width];
	CosHeight = new double[Height];
	SinHeight = new double[Height];
	RevsLength = new int[Length];
	RevsWidth = new int[Width];
	RevsHeight = new int[Height];
```

```
	return true;
}

//удаление памяти
bool FFT_3D::DeleteMemory()
{
	//если память не выделена
	if(Length <= 0 && Log2Length <= 0 && Width <= 0 && Log2Width <= 0
&& Height <= 0 && Log2Height <= 0)	//не задан размер
		return true;
	double *Array = NULL;
	Array = Real[0][0];
	delete[] Array;
	Array = Image[0][0];
	delete[] Array;
	Array = NULL;
	for(int i=0;i<Length;i++)
	{
		/*for(int j=0;j<Width;j++)
		{
			delete[] Real[i][j];
			delete[] Image[i][j];
		}*/
		delete[] Real[i];
		delete[] Image[i];
	}
	delete[] Real;
	delete[] Image;
	Real = NULL;
	Image = NULL;
	delete[] CosLength;
	CosLength = NULL;
	delete[] SinLength;
	SinLength = NULL;
	delete[] CosWidth;
	CosWidth = NULL;
	delete[] SinWidth;
	SinWidth = NULL;
	delete[] CosHeight;
	CosHeight = NULL;
	delete[] SinHeight;
	SinHeight = NULL;
	delete[] RevsLength;
	RevsLength = NULL;
```

```
	Length = 0;
	Log2Length = -1;
	Width = 0;
	Log2Width = -1;
	Height = 0;
	Log2Height = -1;
	return true;
}

//очистка данных
bool FFT_3D::Clear()
{
	for(int i=0;i<Length;i++)
		for(int j=0;j<Width;j++)
			for(int k=0;k<Height;k++)
			{
				Real[i][j][k] = 0;
				Image[i][j][k] = 0;
			}
	return true;
}

//реверс индексов массива
bool FFT_3D::ReversInit()
{
	for(int i=0;i<Length;i++)
		RevsLength[i] = revs(i,Log2Length);
	for(int i=0;i<Width;i++)
		RevsWidth[i] = revs(i,Log2Width);
	for(int i=0;i<Height;i++)
		RevsHeight[i] = revs(i,Log2Height);
	return true;
}

//реверс элементов массива
bool FFT_3D::Revers()
{
	double temp;
	for(int i=0;i<Length;i++)
		for(int j=0;j<Width;j++)
			for(int k=0;k<Height;k++)
			{
				temp = Real[i][j][k];
```

```
                        Real[i][j][k] =
Real[RevsLength[i]][RevsWidth[j]][RevsHeight[k]];
                        Real[RevsLength[i]][RevsWidth[j]][RevsHeight[k]] = temp;
                }
        for(int i=0;i<Length;i++)
                for(int j=0;j<Width;j++)
                        for(int k=0;k<Height;k++)
                        {
                                temp = Image[i][j][k];
                                Image[i][j][k] =
Image[RevsLength[i]][RevsWidth[j]][RevsHeight[k]];
                                Image[RevsLength[i]][RevsWidth[j]][RevsHeight[k]] =
temp;
                        }
        return true;
}

//чтение данных из файла, если GenerateData = true, то создаем новые данные
рандомно и пишем их в файл с реверсированием, если rev = true
bool FFT_3D::InputData(char *Param, char *FileName, bool GenerateData, bool
rev)
{
        if(Param == NULL)
        {
                printf("Error! Param is empty!\n");
                return false;
        }
        if(FileName == NULL)
        {
                printf("Error! FileName is empty!\n");
                return false;
        }
        int RANGE_MIN = 0, RANGE_MAX = 255;
        FILE *f;
        f = fopen(Param,"r");
        if(f == NULL)
        {
                printf("Error! File %s can not be open!\n",FileName);
                return false;
        }
        int length = -1, width = -1, heigth = -1;
        fscanf(f,"%d %d %d",&length, &width, &heigth);
        fclose(f);
        if(length != Length || width != Width || heigth != Height)
```

```
{
        printf("Incorrect size!\n");
        return false;
}
double temp;
if(GenerateData == true)
{
        f = fopen(FileName,"w");
        if(f == NULL)
        {
                printf("Error! File %s can not be open!\n",FileName);
                return false;
        }
        //создаем новые данные рандомно и пишем их в файл
        srand( (unsigned)time( NULL ) );
        for(int i=0;i<Length;i++)
        {
                for(int j=0;j<Width;j++)
                {
                        for(int k=0;k<Height;k++)
                        {
                                if(rev == true)
                                {
                                        temp = (((double) rand() / (double)
RAND_MAX) * RANGE_MAX + RANGE_MIN);

        Real[RevsLength[i]][RevsWidth[j]][RevsHeight[k]] = temp;
                                        Image[i][j][k] = 0;
                                        fprintf(f,"%.0f ",temp);
                                }
                                else
                                {
                                        temp = (((double) rand() / (double)
RAND_MAX) * RANGE_MAX + RANGE_MIN);
                                        Real[i][j][k] = temp;
                                        Image[i][j][k] = 0;
                                        fprintf(f,"%.0f ",temp);
                                }
                        }
                        fprintf(f,"\n");
                }
                fprintf(f,"\n");
        }
        fclose(f);
```

```
		return true;
	}
	else	//читаем из файла
	{
		//читаем данные из файла
		f = fopen(FileName,"r");
		if(f == NULL)
		{
			printf("Error! File %s can not be open!\n",FileName);
			return false;
		}
		for(int i=0;i<Length;i++)
			for(int j=0;j<Width;j++)
				for(int k=0;k<Height;k++)
				{
					if(rev == true)
					{
						fscanf(f,"%lf", &temp);

	Real[RevsLength[i]][RevsWidth[j]][RevsHeight[k]]= temp;
						Image[i][j][k] = 0;
					}
					else
					{
						fscanf(f,"%lf", &temp);
						Real[i][j][k] = temp;
						Image[i][j][k] = 0;
					}
				}
		fclose(f);
	}
	return true;
}

//изменение размера входного сигнала
bool FFT_3D::ChangeData(char *Param, char *FileName)
{
	if(FileName == NULL)
	{
		printf("Error! Param is empty!\n");
		return false;
	}
	if(FileName == NULL)
	{
```

```
		printf("Error! FileName is empty!\n");
		return false;
	}

	int length = -1, log2length = -1, width = -1, log2width = -1, height= -1,
log2height = -1;
	int flag = 0;
	while (true)
	{
		flag = 0;
		printf("Input Length as degree 2: ");
		scanf("%d",&length);
		log2length = IsLog2(length);
		if(log2length < 0)
			printf("Incorrect length!\n");
		else
			flag ++;

		printf("Input Width as degree 2: ");
		scanf("%d",&width);
		log2width = IsLog2(width);
		if(log2width < 0)
			printf("Incorrect width!\n");
		else
			flag ++;

		printf("Input Heigth as degree 2: ");
		scanf("%d",&height);
		log2height = IsLog2(height);
		if(log2height < 0)
			printf("Incorrect height!\n");
		else
			flag ++;

		if(flag == 3)
			break;
		else
			printf("please retry:\n");
	}
	DeleteMemory();
	Length = length;
	Log2Length = log2length;
	Width = width;
	log2width = log2width;
```

```
	Height = height;
	Log2Height = log2height;
	CreateMemory();
	FILE *f;
	f = fopen(Param,"w");
	if(f == NULL)
	{
		printf("Error! File %s can not be open!\n",Param);
		return false;
	}
	fprintf(f,"%d %d %d %d %d %d\n",&Length,&Log2Length, &Width,
&Log2Width, &Height, &Log2Height);
	fclose(f);

	int RANGE_MIN = 0, RANGE_MAX  = 255;
	srand( (unsigned)time( NULL ) );
	f = fopen(FileName,"w");
	if(f == NULL)
	{
		printf("Error! File %s can not be open!\n",FileName);
		return false;
	}
	for(int i=0;i<Length;i++)
	{
		for(int j=0;j<Width;j++)
		{
			for(int k=0;k<Height;k++)
			{
				Real[i][j][k] = (((double) rand() / (double) RAND_MAX) *
RANGE_MAX + RANGE_MIN);
				Image[i][j][k] = 0;
				fprintf(f,"%.0f ",Real[i][j][k]);
			}
			fprintf(f,"\n");
		}
		fprintf(f,"\n");
	}
	fclose(f);
	return true;
}

//запись результатов в файл
bool FFT_3D::OutputData(char *FileName)
{
```

```
	if(FileName == NULL)
	{
		printf("Error! FileName is empty!\n");
		return false;
	}
	FILE *f;
	f = fopen(FileName,"w");
	if(f == NULL)
	{
		printf("Error! File %s can not be open!\n",FileName);
		return false;
	}
	for(int i=0;i<Length;i++)
	{
		for(int j=0;j<Width;j++)
		{
			for(int k=0;k<Height;k++)
			{
				if(Image[i][j][k] == 0)
					fprintf(f,"%.2f ", Real[i][j][k]);
				else
					if(Image[i][j][k] < 0)
						fprintf(f,"%.2f%.2fi ", Real[i][j][k],
Image[i][j][k]);
					else
						fprintf(f,"%.2f+%.2fi ", Real[i][j][k],
Image[i][j][k]);
			}
			fprintf(f,"\n");
		}
		fprintf(f,"\n");
	}
	fclose(f);
	return true;
}

//вычисление комплексной экспоненты
bool FFT_3D::CalcExponent()
{
	const double M_PI = acos(-1.0);
	double TempCos;
	double TempSin;
	//заполняем массивы значениями 2*Pi*i/N, i=[0,N-1]
	if(Length < 1 || Width < 1 || Height < 1)
```

```
{
        printf("Incorrect size!\n");
        return false;
}
if(Length == 1)
{
        CosLength[0] = 1.0;
        SinLength[0] = 0;
        //return true;
}
if(Length == 2)
{
        CosLength[0] = 1.0;
        SinLength[0] = 0;
        CosLength[1] = -1.0;
        SinLength[1] = 0;
        //return true;
}
if(Length == 4)
{
        CosLength[0] = 1.0;             //0
        SinLength[0] = 0;
        CosLength[1] = 0;               //Pi/2
        SinLength[1] = 1.0;
        CosLength[2] = -1.0;            //Pi
        SinLength[2] = 0;
        CosLength[3] = 0;               //3Pi/2
        SinLength[3] = -1.0;
        //return true;
}
TempCos = cos(2*M_PI/Length);     //значение на которое надо повернуть
комплексное число
TempSin = sin(2*M_PI/Length);
for(int i=0;i<Length/4;i++)
{

        CosLength[i] = cos(2*i*M_PI/Length);
        SinLength[i] = sin(2*i*M_PI/Length);
}
for(int i=0;i<Length/4;i++)
{
        CosLength[i + Length/4] = - SinLength[i];
        SinLength[i + Length/4] = CosLength[i];
        CosLength[i + Length/2] = - CosLength[i];
```

```
		SinLength[i + Length/2] = - SinLength[i];
		CosLength[i + 3*Length/4] = SinLength[i];
		SinLength[i + 3*Length/4] = - CosLength[i];
	}

	if(Width == 1)
	{
		CosWidth[0] = 1.0;
		SinWidth[0] = 0;
		//return true;
	}
	if(Width == 2)
	{
		CosWidth[0] = 1.0;
		SinWidth[0] = 0;
		CosWidth[1] = -1.0;
		SinWidth[1] = 0;
		//return true;
	}
	if(Width == 4)
	{
		CosWidth[0] = 1.0;		//0
		SinWidth[0] = 0;
		CosWidth[1] = 0;		//Pi/2
		SinWidth[1] = 1.0;
		CosWidth[2] = -1.0;		//Pi
		SinWidth[2] = 0;
		CosWidth[3] = 0;		//3Pi/2
		SinWidth[3] = -1.0;
		//return true;
	}
	TempCos = cos(2*M_PI/Width);	//значение на которое надо повернуть
комплексное число
	TempSin = sin(2*M_PI/Width);
	for(int i=0;i<Width/4;i++)
	{
		if(i%64 == 0)	//корректировка точности: каждое 64 значение
вычисляем напрямую
		{
			CosWidth[i] = cos(2*M_PI*i/Width);
			SinWidth[i] = sin(2*M_PI*i/Width);
		}
		else		//остальные поворотом предыдущего угла на
единичный
```

```
            {
                CosWidth[i] = CosWidth[i-1]*TempCos - SinWidth[i-
1]*TempSin;
                SinWidth[i] = SinWidth[i-1]*TempCos + CosWidth[i-
1]*TempSin;
            }
        }
        for(int i=0;i<Width/4;i++)
        {
            CosWidth[i + Width/4] = - SinWidth[i];
            SinWidth[i + Width/4] = CosWidth[i];
            CosWidth[i + Width/2] = - CosWidth[i];
            SinWidth[i + Width/2] = - SinWidth[i];
            CosWidth[i + 3*Width/4] = SinWidth[i];
            SinWidth[i + 3*Width/4] = - CosWidth[i];
        }

        if(Height == 1)
        {
            CosHeight[0] = 1.0;
            SinHeight[0] = 0;
            //return true;
        }
        if(Height == 2)
        {
            CosHeight[0] = 1.0;
            SinHeight[0] = 0;
            CosHeight[1] = -1.0;
            SinHeight[1] = 0;
            //return true;
        }
        if(Height == 4)
        {
            CosHeight[0] = 1.0;             //0
            SinHeight[0] = 0;
            CosHeight[1] = 0;               //Pi/2
            SinHeight[1] = 1.0;
            CosHeight[2] = -1.0;            //Pi
            SinHeight[2] = 0;
            CosHeight[3] = 0;               //3Pi/2
            SinHeight[3] = -1.0;
            //return true;
        }
```

```
	TempCos = cos(2*M_PI/Height);	//значение на которое надо повернуть
комплексное число
	TempSin = sin(2*M_PI/Height);
	for(int i=0;i<Height/4;i++)
	{
		if(i%64 == 0)	//корректировка точности: каждое 64 значение
вычисляем напрямую
		{
			CosHeight[i] = cos(2*M_PI*i/Height);
			SinHeight[i] = sin(2*M_PI*i/Height);
		}
		else		//остальные поворотом предыдущего угла на
единичный
		{
			CosHeight[i] = CosHeight[i-1]*TempCos - SinHeight[i-
1]*TempSin;
			SinHeight[i] = SinHeight[i-1]*TempCos + CosHeight[i-
1]*TempSin;
		}
	}
	for(int i=0;i<Height/4;i++)
	{
		CosHeight[i + Height/4] = - SinHeight[i];
		SinHeight[i + Height/4] = CosHeight[i];
		CosHeight[i + Height/2] = - CosHeight[i];
		SinHeight[i + Height/2] = - SinHeight[i];
		CosHeight[i + 3*Height/4] = SinHeight[i];
		SinHeight[i + 3*Height/4] = - CosHeight[i];
	}

	return true;
}

//прямое вычисление преобразования Фурье
bool FFT_3D::DFT(bool direct)
{
	int argx = 0, argy = 0, argz = 0;
	double ***RealData = NULL, ***ImageData = NULL, ***TempReal =
NULL, ***TempImage = NULL, *RealArray = NULL, *ImageArray;
	//выделяем память под результат преобразования Фурье
	RealArray = new double [Length*Width*Height];
	ImageArray = new double [Length*Width*Height];
	RealData = new double**[Length];
	ImageData = new double**[Length];
```

```
	for(int i=0;i<Length;i++)
	{
		RealData[i] = new double*[Width];
		ImageData[i] = new double*[Width];
		for(int j=0;j<Width;j++)
		{
			RealData[i][j] = &RealArray[i*Width*Length + j*Width];
			ImageData[i][j] = &ImageArray[i*Width*Length + j*Width];
		}
	}
	//меняем местами массивы исходный и результирующий, чтобы результат
писался в данные класса
	TempReal = RealData;
	TempImage = ImageData;
	RealData = Real;
	ImageData = Image;
	Real = TempReal;
	Image = TempImage;
	//заполняем значение 0
	Clear();
	int oper;
	if(direct == true)
		oper = -1;
	else
		oper = 1;
	for(int i=0;i<Length;i++)
		for(int j=0;j<Width;j++)
			for(int k=0;k<Height;k++)
			{
				for(int ii=0;ii<Length;ii++)
					for(int jj=0;jj<Width;jj++)
						for(int kk=0;kk<Height;kk++)
						{
							//подсчитываем показатель
экспоненты
							argx = (2*i*ii)%Length;
							argy = (2*j*jj)%Width;
							argz = (2*k*kk)%Height;
							Real[i][j][k] +=
CosLength[argx]*CosWidth[argy]*CosHeight[argz]*RealData[ii][jj][kk] -
oper*SinLength[argx]*SinWidth[argy]*SinHeight[argz]*ImageData[ii][jj][kk];
							Image[i][j][k] +=
CosLength[argx]*CosWidth[argy]*CosHeight[argz]*ImageData[ii][jj][kk] +
oper*SinLength[argx]*SinWidth[argy]*SinHeight[argz]*RealData[ii][jj][kk];
```

```
                                        }
                        }

        double Norm = (double)Length*Width*Height;
        if(direct == false)
        {
                for(int i=0; i<Length; i++)
                        for(int j=0;j<Width;j++)
                                for(int k=0;k<Height;k++)
                                {
                                        Real[i][j][k] /= Norm;
                                        Image[i][j][k] /= Norm;
                                }
        }

        return true;
}

//БПФ Кули-Тьюки
bool FFT_3D::FFT(bool direct)
{
        int x1,x2,y1,y2,z1,z2, oper, argx,argy,argz;
        double Exp_x1y1z1_Real, Exp_x1y1z1_Image, Exp_x2y1z1_Real,
Exp_x2y1z1_Image,
                Exp_x1y2z1_Real, Exp_x1y2z1_Image, Exp_x2y2z1_Real,
Exp_x2y2z1_Image,
                Exp_x1y1z2_Real, Exp_x1y1z2_Image, Exp_x2y1z2_Real,
Exp_x2y1z2_Image,
                Exp_x1y2z2_Real, Exp_x1y2z2_Image, Exp_x2y2z2_Real,
Exp_x2y2z2_Image,
                CosXY, SinXY, CosXZ, SinXZ,
                CosYZ, SinYZ, CosXYZ, SinXYZ;
        int vx, Nvx = Length, dvx = 1, dv2x,px,lx,
                vy, Nvy = Width,  dvy = 1, dv2y,py,ly,
                vz, Nvz = Height, dvz = 1, dv2z,pz,lz;
        if(direct == true)
                oper = 1;
        else
                oper = -1;
        dvx = 1;
        dv2x = dvx*2;
        Nvx = Length/dv2x;
        dvy = 1;
        dv2y = dvy*2;
```

```
Nvy = Width/dv2y;
dvz = 1;
dv2z = dvz*2;
Nvz = Height/dv2z;
for(vx = 1; vx <= Log2Length; vx++)
{
        for(vy = 1; vy <= Log2Width; vy++)
        {
                for(vz = 1; vz <= Log2Height; vz++)
                {
                        for(px = 0; px < Nvx; px++)
                                for(lx = 0; lx < dvx; lx++)
                                        {
                                                x1 = lx + px * dv2x;
                                                x2 = x1 + dvx;
                                                for(py = 0; py < Nvy; py++)
                                                        for(ly = 0; ly < dvy; ly++)
                                                        {
                                                                y1 = ly + py * dv2y;
                                                                y2 = y1 + dvy;

                                                                for(pz = 0; pz < Nvz; pz++)
                                                                        for(lz = 0; lz < dvz;
lz++)
                                                                        {
                                                                                z1 = lz + pz *
dv2z;
                                                                                z2 = z1 + dvz;


Exp_x1y1z1_Real = Real[x1][y1][z1];

Exp_x1y1z1_Image = Image[x1][y1][z1];
                                                                                argx = (Length /
(2 * dvx)) * lx;

Exp_x2y1z1_Real = CosLength[argx] * Real[x2][y1][z1] - oper *
SinLength[argx] * Image[x2][y1][z1];

Exp_x2y1z1_Image = CosLength[argx] * Image[x2][y1][z1] + oper *
SinLength[argx] * Real[x2][y1][z1];
                                                                                argy = (Width/
(2 * dvy)) * ly;
```

```
Exp_x1y2z1_Real = CosWidth[argy]*Real[x1][y2][z1] -
oper*SinWidth[argy]*Image[x1][y2][z1];

Exp_x1y2z1_Image = CosWidth[argy]*Image[x1][y2][z1] +
oper*SinWidth[argy]*Real[x1][y2][z1];
                                                    CosXY =
CosLength[argx]*CosWidth[argy] - SinLength[argx]*SinWidth[argy];
                                                    SinXY =
SinLength[argx]*CosWidth[argy] + CosLength[argx]*SinWidth[argy];

Exp_x2y2z1_Real = CosXY*Real[x2][y2][z1] -
oper*SinXY*Image[x2][y2][z1];

Exp_x2y2z1_Image = CosXY*Image[x2][y2][z1] +
oper*SinXY*Real[x2][y2][z1];

                                                    argz = (Height/
(2 * dvz)) * lz;

Exp_x1y1z2_Real = CosHeight[argz]*Real[x1][y1][z2] -
oper*SinHeight[argz]*Image[x1][y1][z2];

Exp_x1y1z2_Image = CosHeight[argz]*Image[x1][y1][z2] +
oper*SinHeight[argz]*Real[x1][y1][z2];
                                                    CosXZ =
CosLength[argx]*CosHeight[argz] - SinLength[argx]*SinHeight[argz];
                                                    SinXZ =
SinLength[argx]*CosHeight[argz] + CosLength[argx]*SinHeight[argz];

Exp_x2y1z2_Real = CosXZ*Real[x2][y1][z2] -
oper*SinXZ*Image[x2][y1][z2];

Exp_x2y1z2_Image = CosXZ*Image[x2][y1][z2] +
oper*SinXZ*Real[x2][y1][z2];
                                                    CosYZ =
CosWidth[argy]*CosHeight[argz] - SinWidth[argy]*SinHeight[argz];
                                                    SinYZ =
SinWidth[argy]*CosHeight[argz] + CosWidth[argy]*SinHeight[argz];

Exp_x1y2z2_Real = CosYZ * Real[x1][y2][z2] - oper * SinYZ *
Image[x1][y2][z2];
```

```
Exp_x1y2z2_Image = CosYZ * Image[x1][y2][z2] + oper * SinYZ *
Real[x1][y2][z2];
                                                                CosXYZ =
CosLength[argx]*CosYZ - SinLength[argx]*SinYZ;
                                                                SinXYZ =
SinLength[argx]*CosYZ + CosLength[argx]*SinYZ;

Exp_x2y2z2_Real = CosXYZ * Real[x2][y2][z2] - oper * SinXYZ *
Image[x2][y2][z2];

Exp_x2y2z2_Image = CosXYZ * Image[x2][y2][z2] + oper * SinXYZ *
Real[x2][y2][z2];



Real[x1][y1][z1] = Exp_x1y1z1_Real + Exp_x2y1z1_Real +
Exp_x1y2z1_Real + Exp_x2y2z1_Real
                                                                +
Exp_x1y1z2_Real + Exp_x2y1z2_Real + Exp_x1y2z2_Real + Exp_x2y2z2_Real;

Image[x1][y1][z1] = -(Exp_x1y1z1_Image + Exp_x2y1z1_Image +
Exp_x1y2z1_Image + Exp_x2y2z1_Image
                                                                +
Exp_x1y1z2_Image + Exp_x2y1z2_Image + Exp_x1y2z2_Image +
Exp_x2y2z2_Image);


Real[x2][y1][z1] = Exp_x1y1z1_Real - Exp_x2y1z1_Real +
Exp_x1y2z1_Real - Exp_x2y2z1_Real
                                                                +
Exp_x1y1z2_Real - Exp_x2y1z2_Real + Exp_x1y2z2_Real - Exp_x2y2z2_Real;

Image[x2][y1][z1] = -(Exp_x1y1z1_Image - Exp_x2y1z1_Image +
Exp_x1y2z1_Image - Exp_x2y2z1_Image
                                                                +
Exp_x1y1z2_Image - Exp_x2y1z2_Image + Exp_x1y2z2_Image -
Exp_x2y2z2_Image);


Real[x1][y2][z1] = Exp_x1y1z1_Real + Exp_x2y1z1_Real -
Exp_x1y2z1_Real - Exp_x2y2z1_Real
                                                                +
Exp_x1y1z2_Real + Exp_x2y1z2_Real - Exp_x1y2z2_Real - Exp_x2y2z2_Real;
```

```
Image[x1][y2][z1] = -(Exp_x1y1z1_Image + Exp_x2y1z1_Image -
Exp_x1y2z1_Image - Exp_x2y2z1_Image
                                                                +
Exp_x1y1z2_Image + Exp_x2y1z2_Image - Exp_x1y2z2_Image -
Exp_x2y2z2_Image);


Real[x2][y2][z1] = Exp_x1y1z1_Real - Exp_x2y1z1_Real - Exp_x1y2z1_Real
+ Exp_x2y2z1_Real
                                                                +
Exp_x1y1z2_Real - Exp_x2y1z2_Real - Exp_x1y2z2_Real + Exp_x2y2z2_Real;

Image[x2][y2][z1] = -(Exp_x1y1z1_Image - Exp_x2y1z1_Image -
Exp_x1y2z1_Image + Exp_x2y2z1_Image
                                                                +
Exp_x1y1z2_Image - Exp_x2y1z2_Image - Exp_x1y2z2_Image +
Exp_x2y2z2_Image);


Real[x1][y1][z2] = Exp_x1y1z1_Real + Exp_x2y1z1_Real +
Exp_x1y2z1_Real + Exp_x2y2z1_Real
                                                                -
Exp_x1y1z2_Real - Exp_x2y1z2_Real - Exp_x1y2z2_Real - Exp_x2y2z2_Real;

Image[x1][y1][z2] = -(Exp_x1y1z1_Image + Exp_x2y1z1_Image +
Exp_x1y2z1_Image + Exp_x2y2z1_Image
                                                                -
Exp_x1y1z2_Image - Exp_x2y1z2_Image - Exp_x1y2z2_Image -
Exp_x2y2z2_Image);


Real[x2][y1][z2] = Exp_x1y1z1_Real - Exp_x2y1z1_Real +
Exp_x1y2z1_Real - Exp_x2y2z1_Real
                                                                -
Exp_x1y1z2_Real + Exp_x2y1z2_Real - Exp_x1y2z2_Real + Exp_x2y2z2_Real;

Image[x2][y1][z2] = -(Exp_x1y1z1_Image - Exp_x2y1z1_Image +
Exp_x1y2z1_Image - Exp_x2y2z1_Image
                                                                -
Exp_x1y1z2_Image + Exp_x2y1z2_Image - Exp_x1y2z2_Image +
Exp_x2y2z2_Image);
```

```
	Real[x1][y2][z2] = Exp_x1y1z1_Real + Exp_x2y1z1_Real - 
Exp_x1y2z1_Real - Exp_x2y2z1_Real 
                                                                    - 
Exp_x1y1z2_Real - Exp_x2y1z2_Real + Exp_x1y2z2_Real + Exp_x2y2z2_Real;

	Image[x1][y2][z2] = -(Exp_x1y1z1_Image + Exp_x2y1z1_Image - 
Exp_x1y2z1_Image - Exp_x2y2z1_Image 
                                                                    - 
Exp_x1y1z2_Image - Exp_x2y1z2_Image + Exp_x1y2z2_Image + 
Exp_x2y2z2_Image);


	Real[x2][y2][z2] = Exp_x1y1z1_Real - Exp_x2y1z1_Real - Exp_x1y2z1_Real 
+ Exp_x2y2z1_Real 
                                                                    - 
Exp_x1y1z2_Real + Exp_x2y1z2_Real + Exp_x1y2z2_Real - Exp_x2y2z2_Real;

	Image[x2][y2][z2] = -(Exp_x1y1z1_Image - Exp_x2y1z1_Image - 
Exp_x1y2z1_Image + Exp_x2y2z1_Image 
                                                                    - 
Exp_x1y1z2_Image + Exp_x2y1z2_Image + Exp_x1y2z2_Image - 
Exp_x2y2z2_Image);

                                                             //
                                                        }
                                               }
                                 }
                            dvx *= 2;
                            dvy *= 2;
                            dvz *= 2;
                            dv2x *= 2;
                            dv2y *= 2;
                            dv2z *= 2;
                            Nvx /= 2;
                            Nvy /= 2;
                            Nvz /= 2;
                      }
                 }
	}
	//double Norm = 
sqrt((double)Length)*sqrt((double)Width)*sqrt((double)Height);
	double Norm = (double)Length*Width*Height;
	if(direct == false)
```

```
	{
		for(int i=0; i<Length; i++)
			for(int j=0;j<Width;j++)
				for(int k=0;k<Height;k++)
				{
					Real[i][j][k] /= Norm;
					Image[i][j][k] /= Norm;
				}
	}
	return true;
}

//БПФ при помощи комбинации одномерных БПФ
bool FFT_3D::FFTper1D(bool direct)
{
	double temp;
	FFT_1D *f;
	f = new FFT_1D();
	f->Init(Length);
	f->CreateMemory();
	f->CalcExponent();
	//(x,y,z)
	for(int i=0;i<Length;i++)
		for(int j=0;j<Width;j++)
		{
			f->SetData(Real[i][j],Image[i][j]);
			f->FFT(direct);
			f->GetData(Real[i][j],Image[i][j]);
		}
	//меняем местами y и z (транспонируем)
	for(int i=0;i<Length;i++)
		for(int j=0;j<Width;j++)
			for(int k=0;k<j;k++)
			{
				temp = Real[i][j][k];
				Real[i][j][k] = Real[i][k][j];
				Real[i][k][j] = temp;
				temp = Image[i][j][k];
				Image[i][j][k] = Image[i][k][j];
				Image[i][k][j] = temp;
			}
	//(x,z,y)
	for(int i=0;i<Length;i++)
		for(int j=0;j<Width;j++)
```

```
        {
                f->SetData(Real[i][j],Image[i][j]);
                f->FFT(direct);
                f->GetData(Real[i][j],Image[i][j]);
        }
//меняем обратно y и z
for(int i=0;i<Length;i++)
        for(int j=0;j<Width;j++)
                for(int k=0;k<j;k++)
                {
                        temp = Real[i][j][k];
                        Real[i][j][k] = Real[i][k][j];
                        Real[i][k][j] = temp;
                        temp = Image[i][j][k];
                        Image[i][j][k] = Image[i][k][j];
                        Image[i][k][j] = temp;
                }
//(x,y,z)
//меняем местами x и z (транспонируем)
for(int j=0;j<Width;j++)
        for(int i=0;i<Length;i++)
                for(int k=0;k<i;k++)
                {
                        temp = Real[i][j][k];
                        Real[i][j][k] = Real[k][j][i];
                        Real[k][j][i] = temp;
                        temp = Image[i][j][k];
                        Image[i][j][k] = Image[k][j][i];
                        Image[k][j][i] = temp;
                }
//(z,y,x)
for(int i=0;i<Length;i++)
        for(int j=0;j<Width;j++)
        {
                f->SetData(Real[i][j],Image[i][j]);
                f->FFT(direct);
                f->GetData(Real[i][j],Image[i][j]);
        }
//теперь делаем обратное транспонирование
//(z,y,x)
//меняем z и x
for(int j=0;j<Width;j++)
        for(int i=0;i<Length;i++)
                for(int k=0;k<i;k++)
```

```
                {
                        temp = Real[i][j][k];
                        Real[i][j][k] = Real[k][j][i];
                        Real[k][j][i] = temp;
                        temp = Image[i][j][k];
                        Image[i][j][k] = Image[k][j][i];
                        Image[k][j][i] = temp;
                }
        //(x,y,z)
        return true;
}
```

Файл main.cpp – ввод и сохранение данных, замер времени, вызов преобразования Фурье.

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <conio.h>

#include "FFT_1D.h"
#include "FFT_3D.h"

using namespace std;

int IsLog2(int N)
{
        int i,j;
        if(N<=0)
                return -1;
        for(j=0,i=1;i<N;i*=2)
                j++;
        if(i == N)
                return j;
        else
                return -1;
}

int Pow2(int k)
{
        if(k < 0)
                return -1;
        int i = 1;
        for(int j=0;j<k;j++)
                i*=2;
```

```
	return i;
}

int revs(int i, int s)
{
	int k;
	if(i == 0 && s == 0)
		return  0;
	if(int(i / 2) == i / 2.0)
		k = revs(int(i / 2), s - 1);
	else
		k = revs(int(i / 2), s - 1) + Pow2(s - 1);
	return k;
}

int main()
{
	char param_1d[] = "param_1d.txt", data_1d[] = "data_1d.txt",
		result_1d[] = "result_1d.txt", result_1d_fft[] = "result_1d_fft.txt",
		result_1d_fftm[] = "result_1d_fftm.txt",
		param_3d[] = "param_3d.txt", data_3d[] = "data_3d.txt",
		result_3d[] = "result_3d.txt", result_3d_fft[] = "result_3d_fft.txt",
		result_3d_fftm[] = "result_3d_fftm.txt", result_3d_fftper1d[] =
"result_3d_fftper1D.txt";
	clock_t begin, end, WorkTime, StartProgram, FinishProgram,
ProgramWorkTime, tt;
	double second = 1.0 ;	//for calc second, 1000 for Win, 1000000 for Linux
	StartProgram = clock();

	tt = clock();
	FFT_3D *fft_3d;
	fft_3d = new FFT_3D();
	fft_3d->Init(param_3d);
	fft_3d->CreateMemory();
	fft_3d->CalcExponent();
	fft_3d->ReversInit();

	//fft_3d->OutputData("temp1.txt");
	fft_3d->InputData(param_3d,data_3d,true,false);  //генерируем новые
данные

	//БПФ при помощи комбинации одномерных БПФ
	fft_3d->InputData(param_3d,data_3d,false,true);
	begin = clock();
```

```
        WorkTime = (begin - tt) / second;
        //printf("memory = %lf\n",double(WorkTime)/1000.0);
        fft_3d->FFTper1D(true);
        //fft_3d->Revers();
        //fft_3d->FFTper1D(false);
        end = clock();
        WorkTime = (end - begin) / second;
        printf("FFTper1D = %lf\n",double(WorkTime)/1000.0);
        fft_3d->OutputData(result_3d_fftper1d);

        //БПФ по аналогу Кули-Тьюки
        fft_3d->InputData(param_3d,data_3d,false,true);
        begin = clock();
        WorkTime = (begin - tt) / second;
        //printf("memory = %lf\n",double(WorkTime)/1000.0);
        fft_3d->FFT(true);
        //fft_3d->Revers();
        //fft_3d->FFT(false);
        end = clock();
        WorkTime = (end - begin) / second;
        printf("FFT = %lf\n",double(WorkTime)/1000.0);
        fft_3d->OutputData(result_3d_fft);

        FinishProgram = clock();
        ProgramWorkTime = (FinishProgram - StartProgram)/second;
        printf("program work: %lf\n", double(ProgramWorkTime)/1000.0);
        getch();
        return 1;
}
```

Файл Param_3D.txt содержит параметры сигнала: длина, ширина, высота.

4 4 4

Файл Data_3D.txt содержит исходный сигнал

223 114 60 133
170 219 243 125
110 109 9 198
72 201 33 66

167 235 232 154
243 220 5 21
69 41 223 197
220 13 62 122

138 234 248 30
162 16 213 222
166 79 209 224
85 122 217 188

60 64 72 166
129 125 27 102
1 129 71 7
107 77 10 113

Файл result_3dfft.txt содержит результат трёхмерного преобразования Фурье.

8122.00 188.00+70.00i -10.00 188.00-70.00i
488.00-534.00i -44.00-486.00i 196.00-238.00i 328.00+378.00i
222.00 -568.00+138.00i -102.00 -568.00-138.00i
488.00+534.00i 328.00-378.00i 196.00+238.00i -44.00+486.00i

-468.00-964.00i 558.00-394.00i -568.00-524.00i 574.00+274.00i
136.00-488.00i -506.00+690.00i 316.00-76.00i 614.00-366.00i
-276.00-532.00i 466.00+1002.00i -124.00+72.00i 70.00-334.00i
128.00+280.00i -158.00-194.00i 32.00+96.00i 566.00-254.00i

1154.00 -400.00+114.00i 166.00 -400.00-114.00i
-336.00-238.00i 396.00-290.00i 352.00-514.00i -700.00+522.00i
-362.00 1224.00-126.00i 114.00 1224.00+126.00i
-336.00+238.00i -700.00-522.00i 352.00+514.00i 396.00+290.00i

-468.00+964.00i 574.00-274.00i -568.00+524.00i 558.00+394.00i
128.00-280.00i 566.00+254.00i 32.00-96.00i -158.00+194.00i
-276.00+532.00i 70.00+334.00i -124.00-72.00i 466.00-1002.00i
136.00+488.00i 614.00+366.00i 316.00+76.00i -506.00-690.00i