Вычислительно-эффективная реализация дискретного преобразования Фурье

А. Ю. Савинков, email: savinkov_a_yu@sc.vsu.ru

Федеральное государственное бюджетное образовательное учреждение высшего образования «Воронежский государственный университет» (ФГБОУ ВО «ВГУ»)

Аннотация. *В данной работе предложена вычислительно-эффективная программная реализация на С++ дискретного преобразования Фурье для выборок произвольного размера.*

Ключевые слова: *DFT, FFT, Chirp Z-transform*

Введение

Дискретное преобразование Фурье (ДПФ) находит широкое применение при анализе и обработке сигналов как при моделировании так и при реализации систем цифровой обработки сигналов. Сложность вычисления ДПФ $O\ N^2$, где N - длина выборки. При больших N использование ДПФ может существенно сказаться на скорости вычислений. Известны алгоритмы быстрого преобразования Фурье (БПФ), например, алгоритм Кули-Тьюки [1, 2] со сложностью $O\ N \cdot \log N$, но такие алгоритмы могут работать только с определенными размерами выборок, например, $2^n, n \in \mathbb{Z}_0$, где $\mathbb{Z}_0$ - множество целых неотрицательных чисел. Для эффективного вычисления ДФТ по выборкам произвольной длины может быть использован чирп-алгоритм Блюстейна [2, 3]. Чирп-алгоритм (chirp Z-transform) как обобщенный метод вычисления ДФТ был предложен в 1968 году [3] и широко описан в литературе, например [2], но все еще относительно мало распространен среди специалистов в цифровой обработке сигналов. Чирп-алгоритм имеет сложность $O\ N \cdot \log N$, но фактически требует выполнения трех БПФ для выборки длиной $L = 2^{\ 1+\log_2 N}$, где $\cdot$ обозначает округление вверх, т.е. L – это удвоенная длина исходной выборки, округленная вверх до ближайшей целой степени двойки. Также требуется выполнить несколько серий комплексных умножений длиной N или L. Поэтому даже при формально одинаковом порядке сложности, фактическое время выполнения чирп-алгоритма примерно на порядок больше времени выполнения традиционного БПФ для той же длины выборки. Поэтому при практической реализации в программах целесообразно использовать традиционные варианты БПФ для тех размеров выборок, которые ими поддерживаются, и переходить к чирп-алгоритму для остальных размеров выборки.

В данной работе предлагается вычислительно-эффективная реализация ДПФ на языке C++, использующая БПФ, если это возможно, и чирп-алгоритм в остальных случаях. Для расширения множества длин выборок, поддерживаемых БПФ, помимо традиционного алгоритма для выборки 2^n реализованы БПФ для выборок $3 \cdot 2^n$ и $5 \cdot 2^n$, где $n \in \mathbb{Z}_0$. Кроме того, используются кэш для хранения предварительно вычисленных справочных таблиц (lookup tables, LUT) значений комплексных экспонент и перестановок, что дополнительно повышает быстродействие. Несмотря на наличие множества готовых реализаций БПФ и ДПФ в составе различных библиотек (например, FFTW), во многих случаях для использования в программах удобнее иметь простую переносимую реализацию в виде единственного файла, не требующую установки в систему дополнительно ПО, поэтому предлагаемая еще одна реализация ДПФ представляется актуальной.

1. Алгоритмы БПФ

Для реализации БПФ обычно используется алгоритм Кули-Тьюки. Основная идея алгоритма состоит в разбиении исходной выборки отсчетов сигнала на несколько выборок меньшего размера, для которых нужно вычислить ДПФ, а затем объединить полученные результаты. Поскольку сложность вычисления ДПФ $O(N^2)$, то суммарная сложность вычисления нескольких ДПФ меньшего размера будет ниже, чем при прямом вычислении исходного ДПФ.

Рассмотрим выборку сигнала, содержащую четное число отсчетов, тогда k-ый отсчет спектра сигнала, $k \in [0, N)$, выражается формулой (1)

$$S_k = \sum_{n=0}^{N-1} s_n \cdot W^{n \cdot k} \quad (1)$$

где $W = \exp\left(-j \cdot \frac{2\pi}{N}\right)$

Далее разделим исходную выборку сигнала на четные и нечетные отсчеты и преобразуем (1) к виду (2)

$$\sum_{n=0}^{N-1} s_n \cdot W^{n \cdot k} = \sum_{n=0}^{\frac{N}{2}-1} s_{2 \cdot n} \cdot W^{2 \cdot n \cdot k} + \sum_{n=0}^{\frac{N}{2}-1} s_{2 \cdot n+1} \cdot W^{(2 \cdot n+1) \cdot k} = \\ = \sum_{n=0}^{\frac{N}{2}-1} s_{2 \cdot n} \cdot W^{2 \cdot n \cdot k} + W^k \cdot \sum_{n=0}^{\frac{N}{2}-1} s_{2 \cdot n+1} \cdot W^{2 \cdot n \cdot k} \quad (2)$$

Заметим теперь, что при последовательном переборе значений k в диапазоне $[0, N)$ будут получены отсчеты комплексной экспоненты

$W^{2\cdot n\cdot k}$ для двух полных периодов. Следовательно, если $l \in \left[0, \frac{N}{2}\right]$, то $\sum_{n=0}^{\frac{N}{2}-1} s_{2\cdot n} \cdot W^{2\cdot n\cdot l} = \sum_{n=0}^{\frac{N}{2}-1} s_{2\cdot n} \cdot W^{2\cdot n\cdot \left(l+\frac{N}{2}\right)}$ и в выражении (2) суммы можно посчитать только для половины индексов k.

Заметим также, что $W^{l+\frac{N}{2}} = -W^{l}$. Действительно, $W^{l+\frac{N}{2}} = \exp\left(-j\cdot\frac{2\pi}{N}\cdot\left(l+\frac{N}{2}\right)\right) = \exp\left(-j\cdot\frac{2\pi}{N}\cdot l\right)\cdot\exp\left(-j\cdot\frac{2\pi}{N}\cdot\frac{N}{2}\right) = \exp\left(-j\cdot\frac{2\pi}{N}\cdot l\right)\cdot\exp\left(-j\cdot\pi\right) = -\exp\left(-j\cdot\frac{2\pi}{N}\cdot l\right) = -W^{l}$.

На рис. 1 приведена блок-схема базового элемента алгоритма БПФ, состоящего в разделении исходной выборки на два фрагмента, независимом вычислении ДПФ для каждого фрагмента и объединении результатов для получения полного спектра исходной выборки.

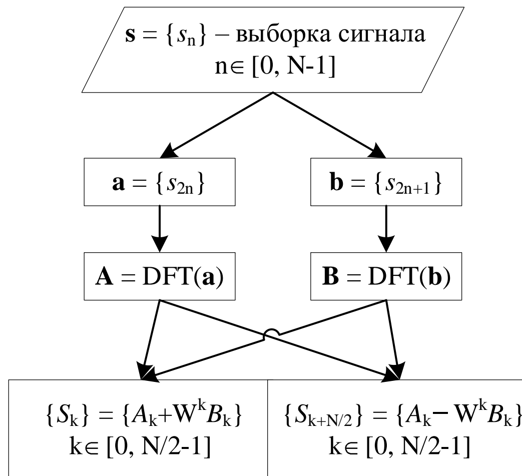

Рис. 1. Базовый элемент алгоритма БПФ

Алгоритм можно рекурсивно применить к выборкам четных и нечетных отсчетов и так далее. Если исходная длина выборки N равна $2^n, n \in \mathbb{Z}_0$, то рекурсивное деление выборок можно продолжить вплоть до выборок из одного отсчета, для которых ДПФ тривиально. При этом результирующая сложность вычислений будет $O\left(N\cdot\log\left(N\right)\right)$.

Если исходная длина выборки $N = k\cdot 2^n, n \in \mathbb{Z}_0$, то рекурсивное деление выборок можно продолжить вплоть до выборок, состоящих из k отсчетов. Оптимизация процедуры вычисления ДПФ для таких выборок

из k отсчетов должна быть выполнена вручную. Например, в [4] приведены алгоритмы вычисления ДПФ для $k = 3$ и $k = 5$.

Фрагмент кода программы для вычисления ДПФ по трем отсчетам сигнала приведен в листинге 1. Исходные данные содержатся в массиве y, результат сохраняется в том же массиве на месте исходных данных.

Листинг 1

```
static const T k = -sin(2.0 * std::numbers::pi / 3.0);
constexpr std::complex<T> J = std::complex<T>(0, 1);
std::complex<T> a = y[1] + y[2];
std::complex<T> t1 = y[0] - a / 2.0;
std::complex<T> t2 = (y[1] - y[2]) * k;
std::complex<T> b = J * t2;
y[0] += a;
y[1] = t1 + b;
y[2] = t1 - b;
```

Код программы для вычисления ДПФ по пяти отсчетам сигнала существенно сложнее и не приводится здесь (полный исходный код для предлагаемой реализации ДПФ доступен по ссылке [5]), отметим только, что в [4] имеется опечатка. Должно быть (обозначения исходной статьи сохранены): $k_{11} = \sin\ 2\pi\ \ 5\ + \sin\ 4\pi\ \ 5$, $k_{12} = -\sin\ 2\pi\ \ 5\ + \sin\ 4\pi\ \ 5$, $B_1 = d_1 + j \cdot d_4$, $B_4 = d_1 - j \cdot d_4$.

Алгоритм БПФ проще всего реализовать с использованием рекурсивных вызовов функций, но такой подход заведомо не оптимальный, прежде всего из-за необходимости копировать фрагменты выборки при ее делении в новые массивы перед рекурсивным вызовом функции вычисления ДПФ. Более эффективное решение состоит в том, чтобы перед вычислением ДПФ переставить элементы исходного массива так, чтобы все последующие вычисления выполнялись бы без рекурсии и копирования данных за несколько последовательных проходов по переупорядоченному массиву исходных данных, при этом результат вычислений заменит исходные данные в массиве.

Например, рассмотрим выборку из 16 элементов, последовательно пронумерованных от 0 до 15. Разделим исходную выборку на две выборки, содержащие четные и нечетные отсчеты исходной выборки соответственно и расположим эти выборки последовательно. Затем полученные выборки разделим еще раз и запишем последовательно уже 4 фрагмента. Продолжим разделения и перестановки, пока не получим выборки из одного элемента, как показано в таблице 1.

Таблица 1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	4	6	8	10	12	14	1	3	5	7	9	11	13	15

0	4	8	12	2	6	10	14	1	5	9	13	3	7	11	15
0	8	4	12	2	10	6	14	1	9	5	13	3	11	7	15
0	8	4	12	2	10	6	14	1	9	5	13	3	11	7	5

Последняя строка таблицы 1 определяет требуемые перестановки (числа в строках таблицы соответствуют индексам элементов в исходном массиве).

Поскольку после последнего разделения получены одноэлементные выборки, то ДФТ для них тривиально – результат просто равен исходным значениям. Поэтому можно сразу попарно объединить результаты одноточечных ДПФ и получить 8 двухэлементных спектров, затем попарно объединить их и т.д., пока не будет получен требуемый 16 элементный спектр исходной выборки.

Перестановка элементов исходного массива сводится к обмену значениями между элементами массива i и j. Обозначим такие обмены значениями как $i \leftrightarrow j$. Для рассматриваемого примера должны быть сделаны следующие обмены, включая тривиальные: $0 \leftrightarrow 0$, $1 \leftrightarrow 8$, $2 \leftrightarrow 4$, $3 \leftrightarrow 12$, $5 \leftrightarrow 10$, $6 \leftrightarrow 6$, $7 \leftrightarrow 14$, $9 \leftrightarrow 9$, $11 \leftrightarrow 13$, $15 \leftrightarrow 15$. При внимательном рассмотрении может быть выявлена закономерность перестановок. Все 16 значений индексов (от 0 до 15) могут быть представлены 4-битными двоичными числами от 0000_2 до 1111_2. При этом в индексах i и j пар перестановок эти двоичные биты всегда записаны в обратном порядке. Действительно $1\ 0001_2 \leftrightarrow 8\ 1000_2$, $2\ 0010_2 \leftrightarrow 4\ 0100_2$, $3\ 0011_2 \leftrightarrow 12\ 1100_2$ и т.д. Такое преобразование числа называется двоично-инверсной преобразованием. Тривиальные перестановки ($0 \leftrightarrow 0$, $6 \leftrightarrow 6$, $9 \leftrightarrow 9$ и $15 \leftrightarrow 15$) также подходят под это правило ввиду симметричности двоичной записи соответствующих значений ($6_{10} = 0110_2$, $9_{10} = 1001_2$), поэтому двоично-инверсное преобразование их не меняет. Многие сигнальные процессоры имеют аппаратную поддержку режима адресации, позволяющего обращаться к элементам массива в соответствии с двоично-инверсным преобразованием индекса. При программной реализации БПФ на процессорах общего назначения двоично-инверсное преобразование может быть сделано явно с использованием битовых операций. На процессорах или в языках программирования, которые не поддерживают явные битовые операции, можно использовать алгоритм Рэйдера [6], который использует только операции сложения, вычитания и деления на два.

Для БПФ длиной $k \cdot 2^n, n \in \mathbb{Z}_0$ двоично-инверсное преобразование должно применяться к номерам блоков из k отсчетов. Блоки

последовательно нумеруются в выходном массиве (после перестановки). Пары индексов для обмена значений вычисляются по формуле (3)

$$\imath + j \cdot \frac{N}{k} \leftrightarrow k \cdot i + j, \qquad i \in \left[0, \frac{N}{k}\right), j \in [0, k) \tag{3}$$

где $N = k \cdot 2^n, n \in \mathbb{Z}_0$; $\imath$ – двоично-инверсное преобразование от i. Но теперь перестановки не будут независимыми. Например, рассмотрим выборку из $N = 6$ элементов и $k = 3$. По формуле (3) получаем следующую цепочку перестановок: $0 \leftrightarrow 0$, $2 \leftrightarrow \mathbf{1}$, $4 \leftrightarrow 2$, $\mathbf{1} \leftrightarrow 3$, … Если просто попарно переставлять элементы входного массива, то перестановки $2 \leftrightarrow \mathbf{1}$ и $\mathbf{1} \leftrightarrow 3$ конфликтуют, и таких конфликтов будет довольно много. Поэтому нужно или использовать второй массив для переставленных значений или не выполнять перестановку вообще, а использовать косвенную индексацию элементов массива через таблицу перестановок.

Предлагаемая реализация ДПФ ориентирована на персональный компьютер и не предназначена для использования во встраиваемом ПО систем цифровой обработки сигналов. В этой связи предполагается наличие достаточного объема свободной памяти и для повышения производительности предпочтительно использование второго массива для переставленных значений.

2. Чирп-алгоритм Блюстейна

В выражении (1) заменим произведение $n \cdot k$ тождественным выражением $\left(n^2 + k^2 - (n-k)^2\right)/2$, тогда

$$S_k = \sum_{n=0}^{N-1} s_n \cdot W^{n \cdot k} = W^{\frac{k^2}{2}} \cdot \sum_{n=0}^{N-1} s_n \cdot W^{\frac{n^2}{2}} \cdot W^{\frac{-(n-k)^2}{2}} \tag{4}$$

Введем обозначения $g_n = s_n \cdot W^{\frac{n^2}{2}}$ и $\nu_n = W^{\frac{n^2}{2}}$, тогда исходя из выражения (4) ДПФ от выборки $\{s_n\}$ может быть вычислено на основе свертки последовательностей $\mathbf{g} = \{g_n\}$ и $\boldsymbol{\nu} = \{\nu_n\}$ (5)

$$\mathbf{c} = \mathbf{g} \otimes \boldsymbol{\nu};\ S_k = W^{\frac{k^2}{2}} \cdot c_k \tag{5}$$

Свертка может быть вычислена на основе теоремы о свертке с использованием БПФ. Поскольку длина последовательностей $\mathbf{g}$ и $\boldsymbol{\nu}$ равна N, то длина их дискретной свертки составит $2 \cdot N - 1$ элементов и для ее вычисления можно использовать БПФ с размером выборки $L = 2^{\lceil \log_2 (2 \cdot N - 1) \rceil}$, т.е. округлить значение $2 \cdot N - 1$ вверх до числа, выражаемого целой степенью двойки. При этом последовательности $\mathbf{g}$ и $\boldsymbol{\nu}$ также должны быть расширены до длины L. Последовательность $\mathbf{g}$ может быть просто дополнена нулями в соответствии с (6)

$$g_n = \begin{cases} s_n \cdot W^{\frac{n^2}{2}}, n < N \\ 0, N \le n < L \end{cases} \quad (6)$$

Поскольку при вычислении свертки $n - k$ может принимать отрицательные значения, последовательность $\nu_n = W^{\frac{n^2}{2}}$ должна быть расширена и в область отрицательных значений n, но поскольку теорема о свертке применима только для периодических последовательностей и в рассматриваемом случае период равен L, то $\nu_{-n} = \nu_{L-n}$ для $n \ne 0$, тогда

$$\nu_n = \begin{cases} W^{\frac{n^2}{2}}, n < N \\ 0, N \le n \le L - N \\ W^{\frac{(L-n)^2}{2}}, L - N < n < L \end{cases} \quad (7)$$

Окончательно получаем следующую последовательность действий для вычисления ДПФ произвольного размера:

- вычислить $L = 2^{\lceil \log_2 (2\cdot N-1) \rceil}$;
- сформировать расширенные последовательности g_n и ν_n по формулам (6) и (7) соответственно;
- вычислить $\mathbf{r} = \mathrm{IFFT}_L(\mathrm{FFT}_L(\mathbf{g}) \cdot \mathrm{FFT}_L(\boldsymbol{\nu}))$;
- вычислить значения отсчетов спектра $S_k = r_k \cdot W^{\frac{k^2}{2}}, k \in [0, N)$.

3. Вычисление обратного ДПФ

Обратное ДПФ может быть вычислено через прямое ДПФ по формуле (8)

$$IDFT(\mathbf{S}) = \frac{1}{N} \cdot \mathrm{conj}(DFT(\mathrm{conj}(\mathbf{S}))) \quad (8)$$

где функция $\mathrm{conj}(\cdot)$ означает комплексное сопряжение. Действительно, по определению обратного ДПФ

$$x_k = \frac{1}{N} \cdot \sum_{n=0}^{N-1} S_n \cdot (W^{n\cdot k})^* = \frac{1}{N} \cdot \left(\sum_{n=0}^{N-1} S_n^* \cdot W^{n\cdot k}\right)^*, \quad k \in [0, N) \quad (9)$$

Таким образом, при наличии реализации прямого ДПФ, всегда можно с минимальными вычислительными издержками получить обратное ДПФ, что и используется в предлагаемом решении.

4. Программная реализация

Исходный код предлагаемой реализации ДПФ доступен по ссылке [5]. Основой реализации является функция fft, определение которой приведено в листинге 2.

Листинг 2

```
template <size_t Radix>
void fft
(
std::vector<std::complex<COMPLEX_TYPE>>& x,
const fft_lut_t<Radix>& lut,
std::function<void(std::complex<COMPLEX_TYPE>*)>
base_transform
)
```

Вектор x при вызове функции содержит отсчеты исходной выборки, а после завершения функции – отсчеты спектра, lut - справочная таблица (будет описана далее), base_transform - базовое - точечное преобразование (k соответсвует параметру шаблона Radix). В реализацию включены базовые преобразования для Radix 2, 3 и 5. При необходимости функции базового преобразования для других Radix могут быть добавлены без изменения кода функции fft.

Параметр шаблона COMPLEX_TYPE может быть float, double или long double и задан через макроопределение DPF_COMPLEX_TYPE, например, #define DPF_COMPLEX_TYPE float. Если макрос DPF_COMPLEX_TYPE не определен, то используется значение double.

Функция fft выполняет двоично-инверсную перестановку входных данных, выполняет базовые преобразования (вызывает base_transform) и последовательно объединяет полученные результаты до получения целевого спектра.

При наличии в компьютере нескольких логических процессоров вычисления могут выполняться параллельно. Поскольку организация параллельных вычислений связана с накладными расходами, выполнять параллельные вычисления имеет смысл только при достаточно большом размере выборки. Порог размера выборки для одного логического процессора, при превышении которого запускаются параллельные вычисления, определяется в зависимости от размерности базового преобразования и может быть задан с использованием макроопределений DFT_RADIX_2_MULTITHREAD_THRESHOLD, DFT_RADIX_3_MULTITHREAD_THRESHOLD и DFT_RADIX_5_MULTITHREAD_THRESHOLD. По умолчанию установлены значения 8192, 6144 и 5120 соответственно.

Как показали измерения, использование справочных таблиц, хранящих заранее вычисленные значения комплексных экспонент и таблицы перестановок, сокращает время выполнения преобразования более, чем в 2 раза. Определение справочной таблицы приведено в листинге 3.

Листинг 3

```
// базовая справочная таблица (lookup table, LUT)
// хранит отсчеты комплексной экспоненты
// используется как для FFT так и для chirp Z-transform
class lut_t
{
public:

    std::vector<std::complex<COMPLEX_TYPE>> w;

    virtual void init(size_t len) = NULL;
};

// справочная таблица для FFT
// добавляет таблицу перестановок и
// данные для параллельных вычислений
template <size_t Radix>
class fft_lut_t : public lut_t
{
public:

    std::vector<size_t> permutations;

    size_t n;

#ifdef USE_MULTITHREAD
    size_t n_cpu;
    size_t per_cpu_n;
    size_t per_cpu_block_size;
#endif

    size_t radix() { return Radix; }
    size_t radix() const { return Radix; }

    // инициализация зависит от Radix
    virtual void init(size_t len);
};

// справочная таблица для ДПФ на основе
// chirp Z-преобразования
class dft_chirp_z_lut_t : public lut_t
{
public:

    virtual void init(size_t len);
};
```

Справочные таблицы хранятся в кэше справочных таблиц, ключом для поиска таблицы в кэше является размерность ДПФ. Для каждого варианта Radix должен использоваться собственный кэш. Реализация кэша приведена в листинге 4.

Листинг 4

```
// кэш справочных таблиц
template <class T>
class lut_cache_t
{
public:

    // возвращает ссылку на справочную таблицу из кэша
    // при отсутствии в кэше нужной таблицы
    // создает новую таблицу и заносит ее в кэш
    const T& get_lut(size_t len)
    {
        auto lut_entry = lut.find(len);
        if (lut_entry == lut.end())
        {
            auto& new_lut = lut[len];
            new_lut.init(len);

            return new_lut;
        }
        return lut_entry->second;
    }

protected:

    std::map<size_t, T> lut;
};
```

Чирп-алгоритм реализуется функцией dft_chirp_z, прототип которой приведен в листинге 5.

Листинг 5

```
void dft_chirp_z
(
std::vector<std::complex<COMPLEX_TYPE>>& x,
const dft_chirp_z_lut_t& lut,
const fft_lut_t<2>& fft_lut
)
```

Поскольку чирп-алгоритм реализуется на основе БПФ, используются две справочные таблицы. Как и при реализации БПФ, при больших выборках и при наличии в компьютере нескольких логических процессоров вычисления в dft_chirp_z выполняются параллельно за счет расщепления циклов. Порог перехода к параллельным вычислениям в dft_chirp_z может быть задан с помощью макроопределения DFT_CHIRP_Z_MULTITHREAD_THRESHOLD (по умолчанию 16384 отсчетов на один логический процессор).

Для удобства использования ДПФ вне зависимости от размерности и типа базового преобразования реализована функция-обертка dft, определение которой приведено в листинге 6.

Листинг 6

```
void dft(std::vector<std::complex<COMPLEX_TYPE>>& x)
```

Функция dft скрывает от пользователя использование кэшей справочных таблиц и выбор типа преобразования. По сути, это одна из двух функций (вторая idft), которые вызывают пользователи в своих программах. Таким образом, использование предлагаемой реализации ДПФ предельно простое: достаточно подключить файл dft.h (#include “dft.h”) и использовать две функции – dft и idft.

5. Оценки производительности

Оценка производительности выполнялась экспериментально на ноутбуке Lenovo IdeaPad S340 с процессором Intel(R) Core(TM) i5-1035G1 (8 логических процессоров) и объемом памяти 8 ГБ. Использовался компилятор MSVC для платформы x64. Среднее время выполнения одного преобразования в зависимости от длины выборки показано на рис. 2.

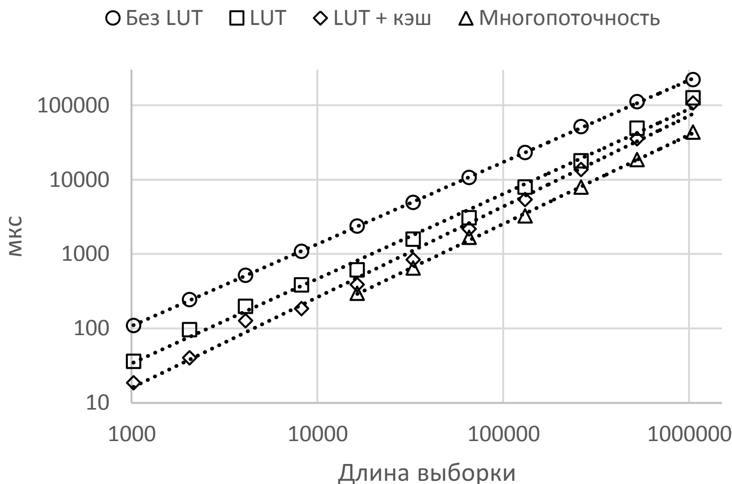

Рис. 2. Среднее время выполнения ДПФ

Измерения подтвердили эффективность каждого из предложенных механизмов повышения производительности: использование справочных таблиц (LUT), использование кэша справочных таблиц и использование параллельных вычислений. Измеренный выигрыш в производительности составил 5-8 раз.

Заключение

Предложена вычислительно-эффективная переносимая реализация прямого и обратного ДПФ на языке C++, не привязанная к аппаратной платформе или операционной системе и не требующая инсталляции дополнительных программных пакетов. Исходный код доступен по ссылке [5]. Измерения подтвердили высокий уровень производительности предложенной реализации.

Список литературы

1. Cooley J.W. An Algorithm for the Machine Calculation of Complex Fourier Series / J.W. Cooley, J.W. Tukey // Mathematics of Computation – 1965 – Vol. 19, No. 90, pp. 297–301.
2. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов: Пер. М.: Мир, 1989. 448 с., ил. с англ.
3. Bluestein L.I. A Linear Filtering Approach to the Computation of Discrete Fourier Transform / L.I. Bluestein // IEEE Transactions on Audio and Electroacoustics – 1970 – Vol. 18, No. 4, pp. 451-455.
4. Löfgren J. On hardware implementation of radix 3 and radix 5 FFT kernels for LTE systems / J. Löfgren, P. Nilsson // 2011 NORCHIP, Lund, Sweden – 2011 – pp. 1-4.
5. Исходный код программы ДПФ [Электронный ресурс] : облачное хранилище – Режим доступа: https://drive.google.com/file/d/14JHTbEXuRUUo0noqvoQooXaumc5AMm1M/view?usp=sharing
6. Д. Рабинер, Б. Гоулд. Теория и применение цифровой обработки сигналов, М.: Мир, 1978, 848 с.