Вычислительно-эффективная реализация дискретного преобразования Фурье

А. Ю. Савинков

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

Цель и актуальность работы

- Цель: реализовать функцию для вычисления ДПФ по выборке произвольного размера со сложностью (( ))
- Актуальность: несмотря на множество существующих реализаций ДПФ (БПФ), все еще нет реализации сочетающей простоту использования и высокую производительность
  - Высокая производительность должна обеспечиваться при любом размере выборки, не обязательно
  - Переносимость (не должно быть зависимости от операционной системы, компилятора, аппаратной платформы)
  - Использование без необходимости установки дополнительных программных пакетов (#include “dft.h” должно быть достаточно)
  - Простой интерфейс в виде вызываемой функции без необходимости предварительного создания каких либо структур данных (контекста)

Алгоритм Кули-Тьюки

Введем обозначение:

ДПФ$_N$

Четные слагаемые

Нечетные слагаемые

ДПФ$_{N/2}$

ДПФ$_{N/2}$

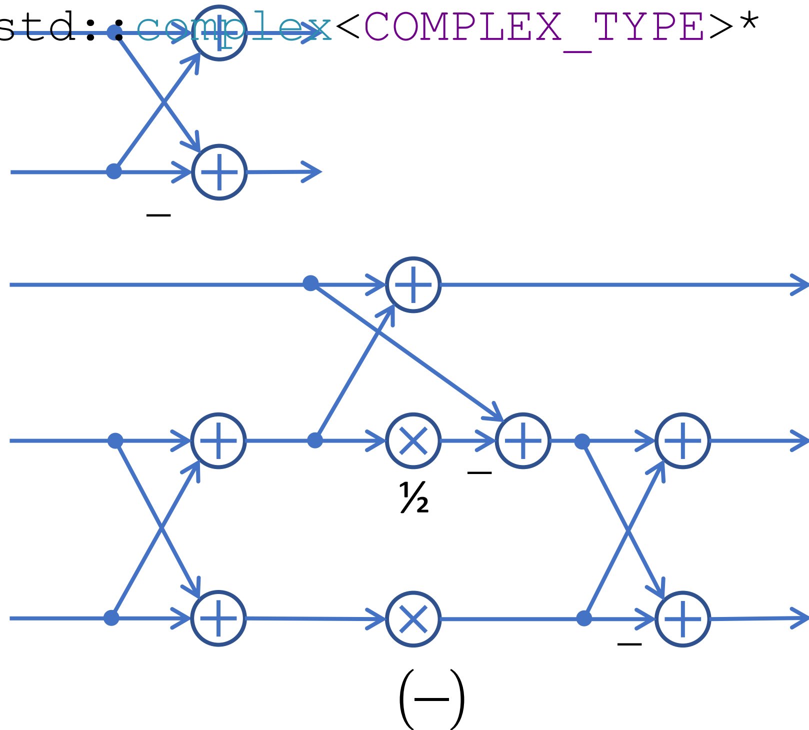
Интерфейс функции БПФ

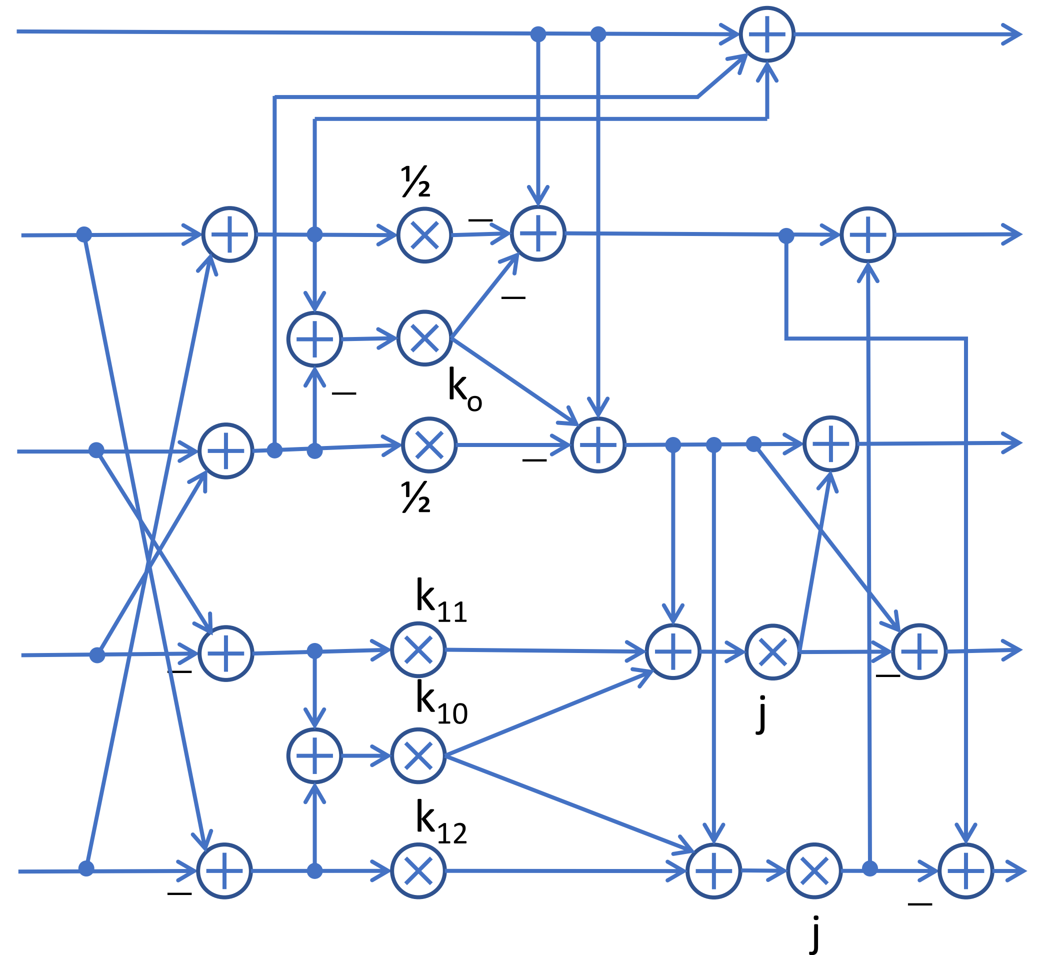
```
template <size_t Radix>
void fft
(
        std::vector<std::complex<COMPLEX_TYPE>>& x,
        const fft_lut_t<Radix>& lut,
        std::function<void(std::complex<COMPLEX_TYPE>*)>
              base_transform /* базовое преобразование для Radix */
);

/* x.size() = Radix * (1 << n) */
#define DFT_COMPLEX_TYPE <type> /* def. double */
```

Базовые ДПФ

```
void fft_radix_2_base_transform(
    std::complex<COMPLEX_TYPE>*
y);
void fft_radix_3_base_transform(
    std::complex<COMPLEX_TYPE>*
y);
void fft_radix_5_base_transform(
    std::complex<COMPLEX_TYPE>*
y);
```





(–) (–)

(–) (–) (–) (–)

Справочные таблицы (LUT)

lut_t

- std::vector<std::complex<COMPLEX_TYPE>> w
- virtual void init(size_t len) = NULL;

dft_chirp_z_lut_t

- virtual void init(size_t len);

fft_lut_t<Radix>

- std::vector<size_t> permutations;
- size_t n; // log2(N/Radix)
- size_t n_cpu;
- size_t per_cpu_n;
- size_t per_cpu_block_size;
- virtual void init(size_t len);

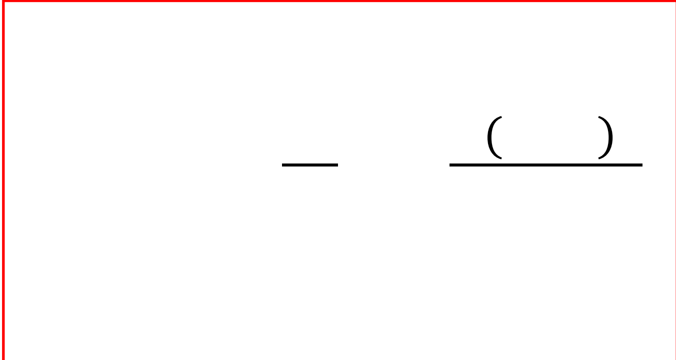
Опционально, только если разрешены параллельные вычисления

Кэш справочных таблиц

```
template <class T>
class lut_cache_t {
public:
    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 Блюстейна (Chirp Z-transform)

ДПФ



Свертка и

Параллельные вычисления

Реализуются на основе std::thread за счет расщепления циклов

```
std::vector<std::thread> transform_threads;
for (size_t n = 0; n < lut.n_cpu; n++) {
    transform_threads.push_back(std::thread(transform_proc,
        &x[n * lut.per_cpu_block_size], lut.per_cpu_block_size));
}
for (size_t n = 0; n < lut.n_cpu; n++) {
    transform_threads[n].join();
}
```

Выполняет БПФ по части входной выборки после двоично-инверсной перестановки, затем полученные фрагменты спектра объединяются на основе алгоритма Кули-Тьюки

Управление параллельными вычислениями

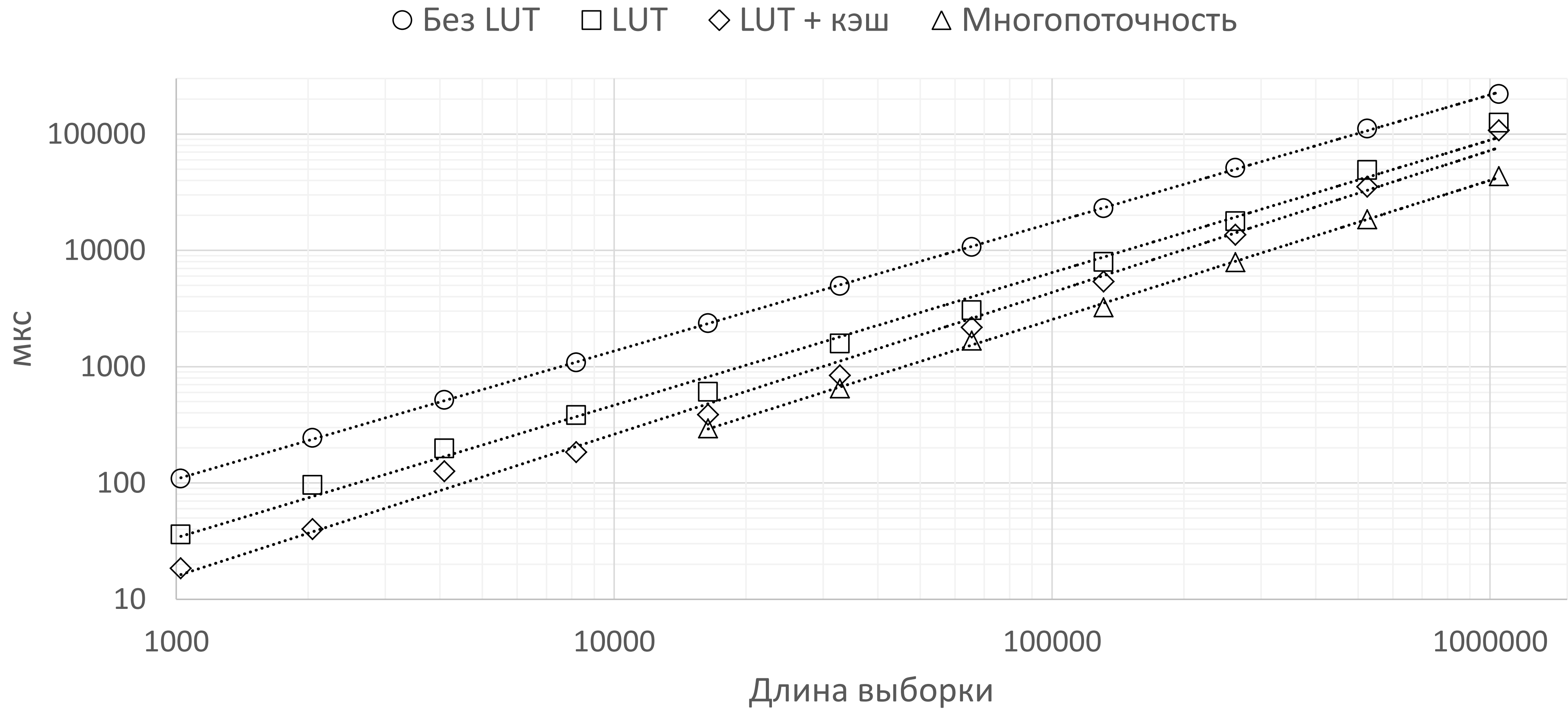
- Число используемых логических процессоров

( )

```
#define DFT_DISABLE_MULTITHREAD /* запрет параллельных вычислений */
#define DFT_RADIX_2_MULTITHREAD_THRESHOLD <N> /* def. 8192 */
#define DFT_RADIX_3_MULTITHREAD_THRESHOLD <N> /* def. 6144 */
#define DFT_RADIX_5_MULTITHREAD_THRESHOLD <N> /* def. 5120 */
#define DFT_CHIRP_Z_MULTITHREAD_THRESHOLD <N> /* def. 16384 */
#define DFT_IDFT_MULTITHREAD_THRESHOLD <N> /* def. 16384 */
```

Обратное ДПФ

Оценка производительности



Пример

```
#include "dft.h"
#include <iostream>
#include <numbers>
int main() {
    try {
        std::vector<std::complex<double>> x(100);
        double f = 2 * std::numbers::pi / x.size();
        for (int i = 0; i < x.size(); i++) {
            x[i] = std::polar(1.0, f * i);
        }
        dft(x);
        for (auto& e : x) std::cout << e << std::endl;
    }
    catch (std::exception& e) {
        std::cout << e.what() << std::endl;
    }
}
```