Marcin Filipek
Wojciech Gomułka

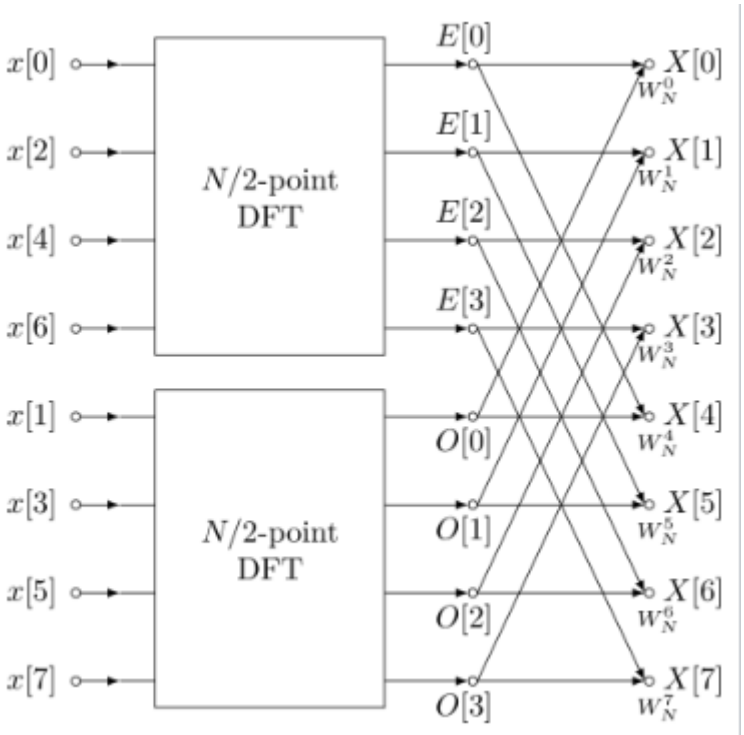
Odszumianie sygnału przy wykorzystaniu szybkiej transformaty Fouriera - dokumentacja

1. Wstęp teoretyczny

Projekt polegał na zaimplementowaniu równoległej wersji szybkiej transformacji Fouriera (ang. *fast Fourier transform, FFT*). Zrównoleglenie przeprowadzono na algorytmie Cooleya-Tukeya - popularnej metodzie umożliwiającej bardziej wydajne (niż bezpośrednio z definicji) obliczanie dyskretnej transformaty Fouriera.

Podejście wprost z definicji charakteryzuje się złożonością rzędu O(n^2), natomiast algorytm Cooleya-Tukeya pozwala zredukować złożoność do O(n log(n)). Wygodnie jest założyć na wejściu, że n jest potęga 2, jednak dostosowanie sygnału do użycia w algorytmie nie jest skomplikowane. Polega bowiem jedynie na dopisaniu zer, aby wyrównać do najbliższej następnej potęgi liczby 2.

Istotą działania algorytmu C-T jest zastosowanie metody “dziel i zwyciężaj”, czyli podziału rekurencyjnego problemu na mniejsze. W przypadku C-T sygnał wejściowy dzielony jest rekurencyjnie na połowy (algorytm radix-2), w taki sposób, że osobno obliczane są wartości transformaty dla próbek sygnału o indeksach parzystych i nieparzystych.



Rysunek 1. Działanie algorytmu C-T (źródło: https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm)

Jak przedstawiono na rysunku 1., operacje przeprowadzane są osobno dla parzystych i nieparzystych indeksów, a następnie scalane ze sobą tak, by poprawnie wyznaczyć dyskretną transformację Fouriera.

Choć algorytm w klasycznej postaci jest rekurencyjny, to istnieje również jego iteracyjna odmiana, potencjalnie prostsza do zrównoleglenia. Poniżej zaprezentowano pseudokody obu podejść (rekurencyjnego i iteracyjnego) na podstawie których zaimplementowane zostały algorytmy w projekcie:

```
X_0,...,N-1 ← ditfft2(x, N, s):          DFT of (x_0, x_s, x_2s, ..., x_(N-1)s):
    if N = 1 then
        X_0 ← x_0                                trivial size-1 DFT base case
    else
        X_0,...,N/2-1 ← ditfft2(x, N/2, 2s)      DFT of (x_0, x_2s, x_4s, ..., x_(N-2)s)
        X_N/2,...,N-1 ← ditfft2(x+s, N/2, 2s)    DFT of (x_s, x_s+2s, x_s+4s, ..., x_(N-1)s)
        for k = 0 to N/2-1 do                    combine DFTs of two halves into full DFT:
            p ← X_k
            q ← exp(-2πi/N k) X_k+N/2
            X_k ← p + q
            X_k+N/2 ← p - q
        end for
    end if
```

Rysunek 2. Pseudokod rekurencyjnego algorytmu C-T

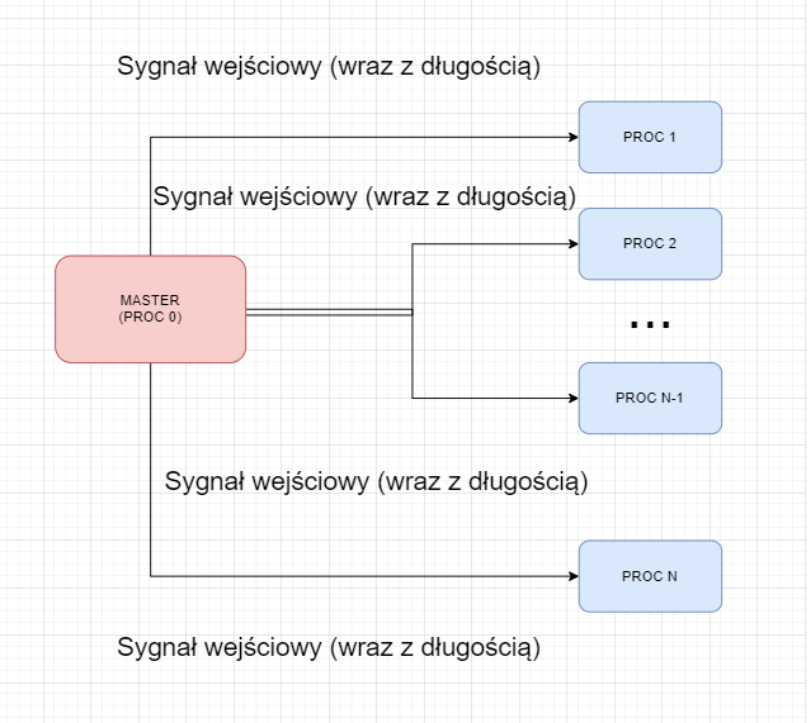
```
algorithm iterative-fft is
    input: Array a of n complex values where n is a power of 2.
    output: Array A the DFT of a.

    bit-reverse-copy(a, A)
    n ← a.length
    for s = 1 to log(n) do
        m ← 2^s
        ω_m ← exp(-2πi/m)
        for k = 0 to n-1 by m do
            ω ← 1
            for j = 0 to m/2 - 1 do
                t ← ω A[k + j + m/2]
                u ← A[k + j]
                A[k + j] ← u + t
                A[k + j + m/2] ← u - t
                ω ← ω ω_m

    return A
```

Rysunek 3. Pseudokod iteracyjnego algorytmu C-T

2. Zrównoleglenie

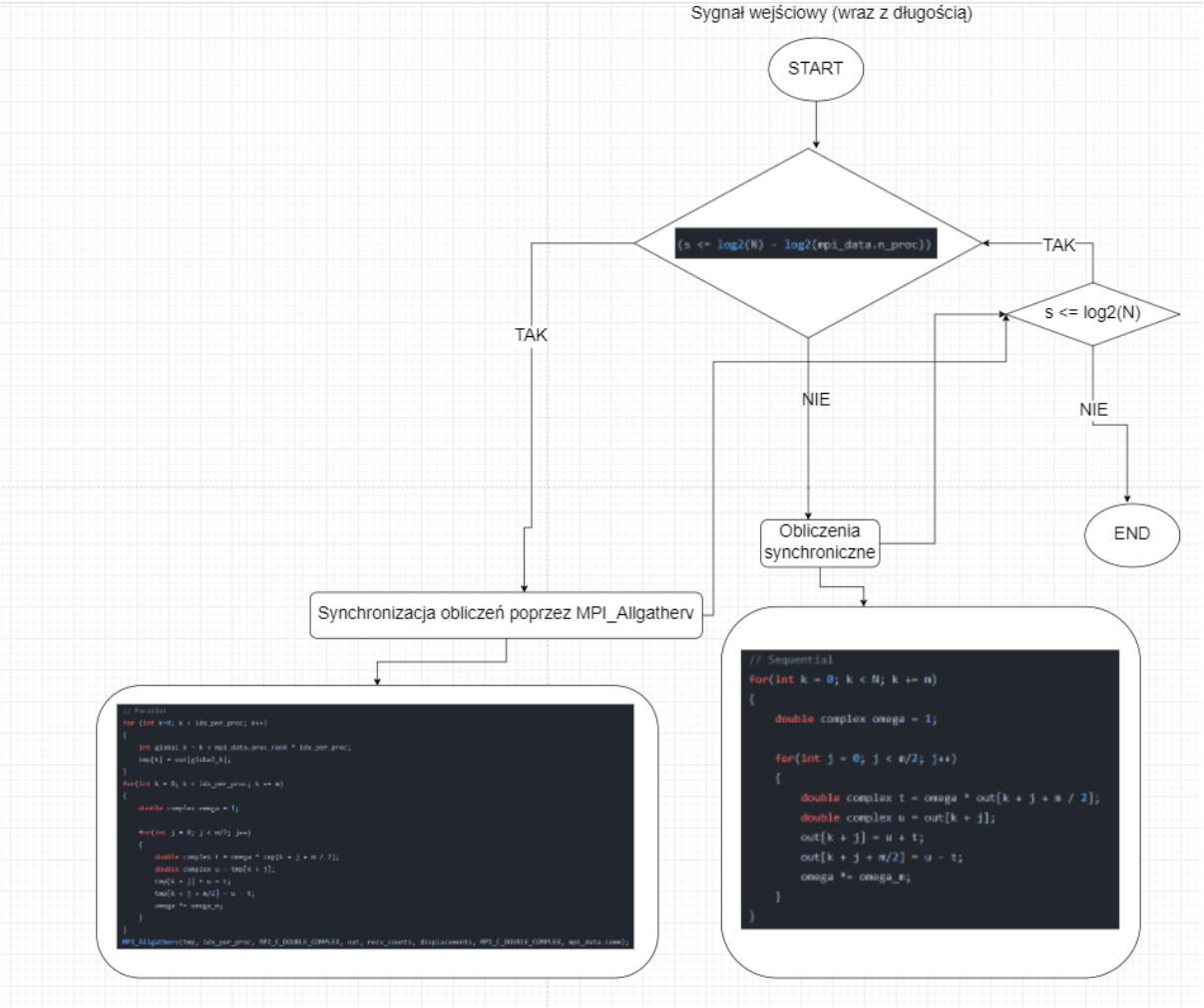
Punkt wyjścia do przygotowania równoległego algorytmu stanowił iteracyjny algorytm Cooleya-Tukeya. Ze względu na sposób przetwarzania danych w algorytmie C-T ("*butterfly operations*" - krzyżowanie widoczne na rysunku 1.) podejście polegające na zwyczajnym podziale tablicy na wiele części przetwarzanych przez różne wątki/procesy nie było wystarczające (choć warto nadmienić, że przy obliczaniu DFT z definicji, prawdopodobnie byłoby możliwe do zaaplikowania). Należało zapewnić przekazywanie danych w odpowiednim momencie do innych procesów, co stanowiło nietrywialne zadanie. Jednym z rozwiązań tego problemu okazała się metoda *MPI_Allgatherv* działająca w sposób podobny do *broadcastu*, ale dodatkowo "sklejająca" dane pochodzące z różnych procesów.

Przed samym przystąpieniem do obliczeń konieczne jest rozpropropagowanie danych wejściowych (w tym przypadku pewnego sygnału) do wszystkich węzłów. Ponadto należy uprzednio podać długość sygnału, aby każdy z procesów przydzielił pamięc o stosownym rozmiarze. Odpowiedzialność tę przejmuje na siebie proces master (ustanowiony jako proces o identyfikatorze 0, a programistyczna realizacja oparta jest o wykorzystanie funkcji *MPI_Bcast*). We wstępie wspomniano, że długość sygnału wejściowego powinna być potęgą liczby 2 lub powinna zostać do takiej długości zmapowana. Podobnie jest z podziałem na procesy. Ich liczba również powinna być potęgą liczby dwa, aby możliwe było zrównoleglenie operacji w algorytmie Radix-2 (podejście naiwne, choć wolniejsze, jest zasadniczo bardziej elastyczne pod tym względem). Broadcast danych do poszczególnych procesów zwizualizowano na rysunku 4.



Rysunek 4. Schematyczna wizualizacja broadcastu do innych węzłów.

Następnym krokiem jest sama realizacja algorytmu równoległego. Zrównoleglone zostały jego operacje począwszy od pierwszej wewnętrznej pętli (*vide* rysunek 1.). Wobec tego wszystkie procesy bez zmian iterują *s od 1 do log2(N).* Następnie algorytm postępuje tak jak pokazano na rysunku 5. Sprawdzane jest czy *s* dotarło do ostatniej iteracji i wówczas następują sekwencyjne obliczenia, w przeciwnym wypadku obliczenia będą synchronizowane między procesami przy pomocy *MPI_Allgetherv*. Istotne są również ograniczenia w indeksach i iteracjach uzależnione od tego, który proces wykonuje kod.



Rysunek 5. Sposób zrównoleglenia w iteracyjnym C-T

Krótka dokumentacja kodu:

void dft_naive(double complex *in, double complex *out, size_t N, MPI_Data mpi_data) - oblicza DFT z definicji, złożoność O(n^2)

void fft_radix2(double complex *in, double complex *out, size_t N, size_t s, MPI_Data mpi_data) - oblicz FFT rekurencyjnie (niewykorzystywana i niezrównoleglona)

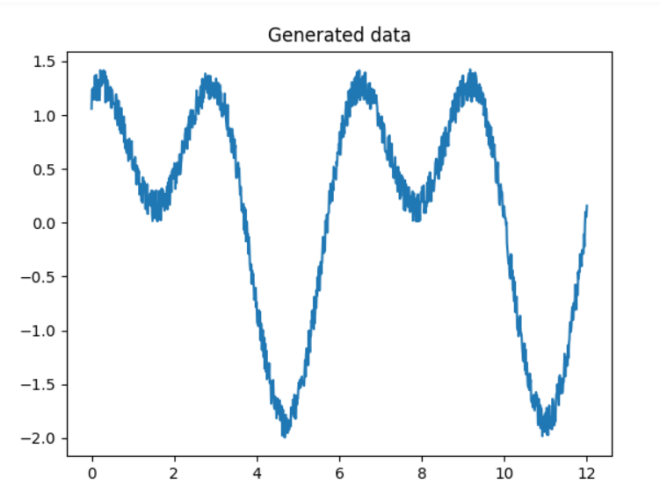
static int rev(unsigned int num, size_t N) - wyznacza tak zwaną odwrotność bitową

static void bit_reversal_copy(double complex* result, double complex* input, size_t N, MPI_Data mpi_data) - mapuje indeksy w tablicy zgodnie z tzw. odwrotnością bitową
void fft_radix2_iter(double complex *in, double complex *out, size_t N, MPI_Data mpi_data) - najważniejsza funkcja w algorytmie - zrównoleglony iteracyjny FFT

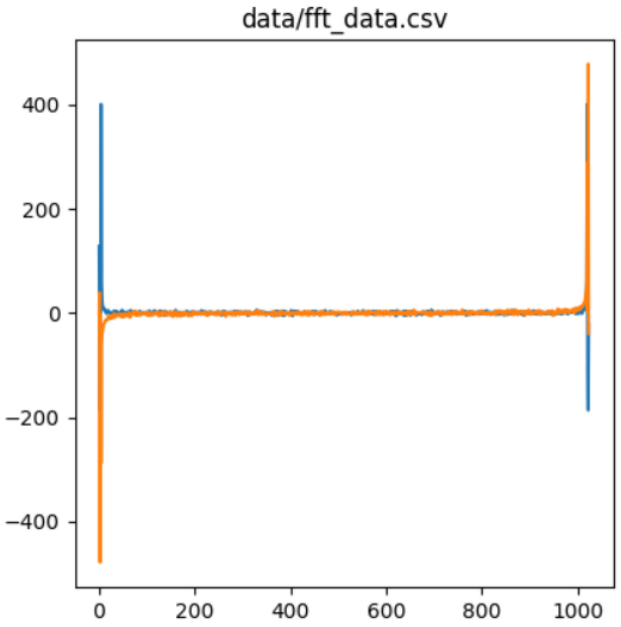
3. Wyniki

Program pozwala na obliczanie DFT oraz odwrotnej DFT metodą zrównoleglonego algorytmu Cooleya-Tukeya. Potencjalnie może posłużyć do odszumiania sygnału (jeśli po przejściu z dziedziny pierwotnej sygnał zostanie poddany dyskryminacji, a następnie tak przekształcony sygnał zostanie przywrócony do dziedziny pierwotnej transformacją odwrotną).

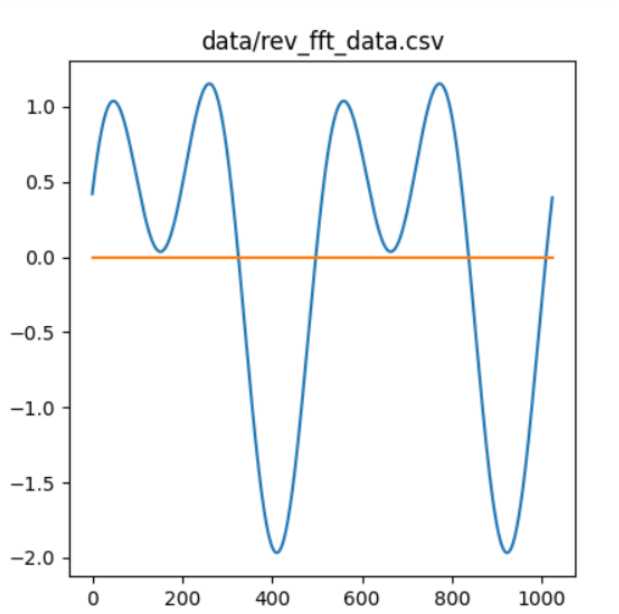
Poniżej na rysunkach 5., 6., 7. zaprezentowano przykładowe przejście z dziedziny pierwotnej do dziedziny transformaty i z powrotem (transformacja odwrotna):



Rysunek 5. Wejściowy sygnał zaszumiony



Rysunek 6. Rezultat działania transformacji Fouriera (na pomarańczowo część zespolona, na niebiesko część rzeczywista)



Rysunek 7. Odszumiony sygnał (działanie transformacji odwrotnej z uprzednim zastosowaniem progu dyskryminacji do usunięcia szumu)

4. Podsumowanie i wnioski

Szybko stało się jasne, że zrównoleglenia algorytmu radix-2 może stanowić trudniejsze wyzwanie, niż zrównoleglenie obliczania transformaty z definicji. Z tego powodu odstąpiono od prób działania na algorytmie rekurencyjnym (de facto najbardziej naturalnym w przypadku obliczeń sekwencyjnych), a przystąpiono do zrównoleglania iteracyjnej odmiany algorytmu C-T.

Najważniejszy problem stanowił podział sygnału w taki sposób, by węzły mogły przetwarzać go równolegle, przy równoczesnym zachowaniu współzależności wynikających ze specyfiki algorytmu. Receptą okazało się użycie metody *MPI_Allgatherv*, która pozwoliła na synchronizację tablic posiadanych w danym momencie przez poszczególne procesy.

Źródła:

https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm - opis algorytmu, główne źródło implementacji i pseudokodów dla FFT
https://www.mpich.org/static/docs/v3.2/ - dokumentacja
https://www.rookiehpc.com/mpi/docs/ - dokumentacja i przykłady