SPRAWOZDANIE LAB13 MARCIN BARANOWSKI

Algorytm FFT (*Fast Fourier Transform*) dla dyskretnej transformacji Fouriera (DFT, *Discrete Fourier Transform*) i jego krewniacy dla wyznaczania dyskretnej transformacji cosinusów (DCT, *Discrete Cosine Transform* i MDCT, *Modified Discrete Cosine Transform*), choć dotyczą pozornie dość abstrakcyjnych zadań, zrewolucjonizowały wiele dziedzin życia. Między innymi wykorzystuje się je w

- kompresji obrazów w formacie JPEG (DCT)
- kompresji dźwięku w formacie MP3 i pokrewnych (MDCT)
- rozwiązywaniu ważnych równań różniczkowych cząstkowych (DFT)

a także do

- filtrowania szumów (DFT)
- szybkiego mnożenia wielomianów(DFT)

W niniejszym wykładzie ograniczymy się do przedstawienia szybkiego algorytmu rozwiązywania zadania DFT. Algorytmy rozwiązywania zadań pokrewnych (DCT, MDCT, itp.) opierają się na podobnych zasadach.

Zacznijmy od postawienia zadania obliczeniowego DFT. Jest ono (pozornie!) banalne i jednocześnie wydumane:

Dla danego zestawu N liczb $f_\alpha \in C, \alpha = 0, \ldots, N-1$, wyznaczyć N wartości

$$c_j = \sum_{\alpha=0}^{N-1} f_\alpha \omega_N^{j\alpha},$$

dla $j = 0, \ldots, N-1$, przy czym $\omega_N = \exp(-i\frac{2\Pi}{N})$. Jak pamiętamy, jednostka urojona i spełnia $i = \sqrt{-1}$. Taką operację nazywamy **dyskretną transformacją Fouriera**, DFT.

Ponieważ dane f są wektorem $f = [f_0, \ldots, f_{N-1}]^T$, wynik c też jest wektorem, a zadanie jest liniowe, możemy wszystko zapisać macierzowo:

$$c = F_N f,$$

gdzie

$$F_N = (\omega_N^{j\alpha})_{j,\alpha=0,\ldots,N-1} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \cdots & \omega_N^{2(N-1)} \\ & & & \cdots & \\ 1 & \omega_N^{N-1} & \omega_N^{2(N_1)} & \cdots & \omega_N^{(N-1)^2} \end{pmatrix}.$$

Tak więc, gdyby naiwnie podejść do zadania, moglibyśmy je rozwiązać brutalnie, tworząc na początek macierz F_N, a następnie wyznaczając iloczyn macierzy przez wektor, co łatwo zrobić kosztem $O(N^2)$ operacji. Dlatego istotne jest, że algorytm FFT, który za chwilę omówimy, będzie działać kosztem $O(N \log N)$, czyli praktycznie liniowym (w obu przypadkach stałe proporcjonalności są niewielkie) w dodatku będzie miał znacznie mniejsze wymagania pamięciowe.

Implementacja kodu:
https://github.com/BradenHarrelson/FastFourierTransform/blob/master/FFT_Parallel.c