Parallel Version n-Dimensional Fast Fourier Transform Algorithm

Analog of the Cooley-Tukey Algorithm

M. V. Noskov and V. S. Tutatchikov

Institute of Space and Information Technology, Siberian Federal University, Kirenskogo Street 26, Krasnoyarsk, Russia

Keywords: Multi-dimensional Discrete Fourier Transform, Cooley-Tukey FFT, Parallel Algorithm.

Abstract: One-, two- and three-dimensional fast Fourier transform (FFT) algorithms has been widely used in digital processing. Multi-dimensional discrete Fourier transform is reduced to a combination of one-dimensional FFT for all coordinates due to the increased complexity and the large amount of computation by increasing the dimensional of the signal. This article provides a general Cooley-Tukey algorithm analog, which requires less complex operations of additional and multiplication than the standard method, and runs 1.5 times faster than analogue in Matlab.

1 INTRODUCTION

One-, two- and three-dimensional fast Fourier transform (FFT) algorithms has been widely used in digital processing (Dudgeon, 1983, Blahut, 1985). Multi-dimensional discrete Fourier transform is reduced to a combination of one-dimensional FFT for all coordinates due to the increased complexity and the large amount of computation by increasing the dimensional of the signal. This article provides a general Cooley-Tukey algorithm analog, which requires less complex operations of additional and multiplication than the standard method (Tutatchikov, 2013). Testing of the resulting algorithm in two- and three-dimensions in comparison with the standard algorithm in Matlab (Gonzalez, 2009).

2 THE ALGORITHM DESCRIPTION

Let us have a look at the signal f, which is an n-dimensional periodic signal with a period 2^s of over all n coordinate with values in a complex space. The counts are given as $f_{x_1,\ldots,x_n} = f(x_1,\ldots,x_n)$, where $x_i, i = 1,\ldots,n$ take values $0,1,\ldots,2^s-1$. The discrete Fourier transformation (DFT) $F_{y_1,\ldots,y_n} = F(y_1,\ldots,y_n)$ for the signal $f(x_1,\ldots,x_n)$ is given in the formula:

$$F(y_1,\ldots,y_n) = \sum_{x_1=0}^{2^s-1}\ldots\sum_{x_n=0}^{2^s-1} f(x_1,\ldots,x_n)\cdot \cdot e^{\frac{2\pi i(x_1y_1+\ldots+x_ny_n)}{2^s}} \quad (1)$$

where $y_i, i = 1,\ldots,n$ take values $0,\ldots,2^s-1$.

2.1 n-Dimensional FFT

Transform the formula (1) as follows:

$$F^1(y_1,\ldots,y_n) = \sum_{b_1=0}^{1}\ldots\sum_{b_n=0}^{1} F^1(y_1^1+2^{s-1}b_1, \ldots, y_n^1+2^{s-1}b_n) = \sum_{b_1=0}^{1}\ldots\sum_{b_n=0}^{1}\cdot\sum_{a_1=0}^{1}\ldots\sum_{a_n=0}^{1}\cdot \cdot(-1)^{b_1+\ldots+b_n}\cdot e^{\frac{2\pi i(y_1^1a_1+\ldots+y_n^1a_n)}{2^s}}\cdot \cdot g^1_{a_1,\ldots,a_n}(y_1^1+2^{s-1}b_1,\ldots,y_n^1+2^{s-1}b_n) \quad (2)$$

where coordinates y_i^1 of the final counts subsignals $g^1_{a_1,\ldots,a_n}$ run 2^{s-1} values, $x_i^1 = 0:2^{s-1}-1$,

Noskov M. and Tutatchikov V..
Parallel Version n-Dimensional Fast Fourier Transform Algorithm - Analog of the Cooley-Tukey Algorithm.
DOI: 10.5220/0005461401140117
In *Proceedings of the 5th International Workshop on Image Mining. Theory and Applications* (IMTA-5-2015), pages 114-117
ISBN: 978-989-758-094-9

$i = 1:n$, F^1 - FFT of source signal f. For convenience, denote $F^0 = f$:

$$F^1(y_1,...,y_n) = \sum_{b_1=0}^{1}...\sum_{b_n=0}^{1} \cdot \sum_{a_1=0}^{1}...\sum_{a_n=0}^{1}(-1)^{b_1+...+b_n} e^{\frac{2\pi i(y_1^1 a_1+...+y_n^1 a_n)}{2^s}} \cdot \sum_{x_1^1=0}^{2^{s-1}-1}...\sum_{x_n^1}^{2^{s-1}-1} F^0(2x_1^1+a_1,...,2x_n^1+a_n) \cdot e^{\frac{2\pi i[(2x_1^1+a_1)(y_1^1+2^{s-1}b_1)+...+(2x_n^1+a_n)(y_n^1+2^{s-1}b_n)]}{2^s}} \quad (3)$$

Continue the same procedure for each $g^1_{a_1,...,a_n}$, that is represented signal $g^1_{a_1,...,a_n}$ as a sum subsignals:

$$g^1_{a_1,...,a_n} = \sum_{\beta} g^2_{\beta_1,...,\beta_n} \quad (4)$$

where coordinates of the final counts subsignals $g^2_{\beta_1,...,\beta_n}$ run 2^{s-2} values.

Continuing this process, we can be represented $F^1(y_1,...,y_n)$ as the sum of DFT signals, wherein each of the n coordinates counts runs on only two values, we obtain the following formula for calculating $F^v(y_1,...,y_n)$:

$$F^v(y_1,...,y_n) = \sum_{c=0}^{2^{v-1}-1}\sum_{b_1=0}^{1}...\sum_{b_n=0}^{1} \cdot \sum_{a_1=0}^{1}...\sum_{a_n=0}^{1}(-1)^{b_1+...+b_n} e^{\frac{2\pi i((y_1^v+2^{s-v+1}c)a_1)}{2^s}} \cdot ... \cdot e^{\frac{2\pi i((y_n^v+2^{s-v+1}c)a_n)}{2^s}} g^v_{a_1,...,a_n}(y_1^v + 2^{s-v}b_1 + 2^{s-v+1}c,...,y_n^v + 2^{s-v}b_n + 2^{s-v+1}c) \quad (5)$$

where $v = 1:s$ - step number of the partition $F(x_1,...,x_n)$ on the subsignals.

Consider in more detail the formula (5):

$$F^v(y_1,...,y_n) = \sum_{c=0}^{2^{v-1}-1}\sum_{b_1=0}^{1}...\sum_{b_n=0}^{1} \cdot F^v(y_1^v + 2^{s-v}b_1 + 2^{s-v+1}c,...,y_n^v + 2^{s-v}b_n + 2^{s-v+1}c) = \sum_{c=0}^{2^{v-1}-1}\sum_{b_1=0}^{1}...\sum_{b_n=0}^{1} \cdot \sum_{a_1=0}^{1}...\sum_{a_n=0}^{1}(-1)^{b_1+...+b_n} e^{\frac{2\pi i((y_1^v+2^{s-v+1}c)a_1)}{2^s}} \cdot ... \cdot e^{\frac{2\pi i((y_n^v+2^{s-v+1}c)a_n)}{2^s}} \sum_{x_1^v=0}^{2^{s-v}-1}...\sum_{x_n^v}^{2^{s-v}-1} \cdot F^{v-1}(2x_1^v + a_1 + 2^{s-v+1}c,...,2x_n^v + a_n + 2^{s-v+1}c) \cdot e^{\frac{2\pi i[(2x_1^v+a_1+2^{s-v+1}c)(y_1^v+2^{s-1}b_1+2^{s-v+1}c)]}{2^s}} \cdot ... \cdot e^{\frac{2\pi i[(2x_n^v+a_n+2^{s-v+1}c)(y_n^v+2^{s-1}b_n+2^{s-v+1}c)]}{2^s}} \quad (6)$$

where $x_i^v, y_i^v = 0:2^{s-v}-1$, $i = 1:n$, $F^s(y_1,...,y_n) = F(y_1,...,y_n)$ - discrete Fourier transformation f.

2.2 Parallel Algorithm FFT

Calculation $F(y_1,...,y_n)$ can be parallelized on independent flows calculations. In the presence $2^q, 0 < q < s$ of flow formula (6) takes the form:

$$F^v(y_1,...,y_n) = \sum_{p=0}^{q}\sum_{t=0}^{2^p-1}\sum_{c=0}^{2^{v-1}-1}\sum_{b_1=0}^{1}...\sum_{b_n=0}^{1} \cdot F^v(y_1^v + 2^{s-v-p}b_1 + 2^{s-v-p+1}c + 2^{s-v-p+2}t, y_2^v + 2^{s-v-p}b_2 + 2^{s-v-p+1}c,...,y_n^v + 2^{s-v}b_n + 2^{s-v+1}c) \quad (7)$$

Consider in more detail the formula (7):

$$F^v(y_1,\dots,y_n)=\sum_{p=1}^{q}\sum_{t=0}^{2^{p-1}-1}\sum_{c=0}^{2^{v-1}-1}\sum_{b_1=0}^{1}\dots\sum_{b_n=0}^{1}\cdot$$
$$\cdot\sum_{a_1=0}^{1}\dots\sum_{a_n=0}^{1}(-1)^{b_1+\dots+b_n}\cdot$$
$$e^{\frac{2\pi i((y_1^v+2^{s-v+1}c+2^{s-v-p+2}t)a_1)}{2^s}}\cdot e^{\frac{2\pi i((y_2^v+2^{s-v+1}c)a_2)}{2^s}}\cdot \qquad (8)$$
$$\cdot\dots\cdot e^{\frac{2\pi i((y_n^v+2^{s-v+1}c)a_n)}{2^s}} g^v_{a_1,\dots,a_n}(y_1^v+2^{s-v}b_1+$$
$$+2^{s-v+1}c+2^{s-v-p+2}t, y_2^v+2^{s-v}b_2+$$
$$2^{s-v+1}c,\dots,y_n^v+2^{s-v}b_n+2^{s-v+1}c)$$

Subsignals $g^v_{a_1,\dots,a_n}$ may be described as follows:

$$F^v(y_1,\dots,y_n)=\sum_{c=0}^{2^{v-1}-1}\sum_{b_1=0}^{1}\dots\sum_{b_n=0}^{1}\cdot$$
$$\cdot\sum_{a_1=0}^{1}\dots\sum_{a_n=0}^{1}(-1)^{b_1+\dots+b_n}\sum_{p=1}^{q}\sum_{t=0}^{2^{p-1}-1}\cdot$$
$$\cdot e^{\frac{2\pi i\left((y_1^v+2^{s-v+1}c+2^{s-v-p+2}t)a_1\right)}{2^s}}\cdot$$
$$\cdot e^{\frac{2\pi i\left((y_2^v+2^{s-v+1}c)a_2\right)}{2^s}}\cdot\dots\cdot e^{\frac{2\pi i\left((y_n^v+2^{s-v+1}c)a_n\right)}{2^s}}\cdot$$
$$\cdot\sum_{p=1}^{q}\sum_{t=0}^{2^{p-1}-1}\sum_{x_1^v=0}^{2^{s-v}-1}\dots\sum_{x_n^v}^{2^{s-v}-1}F^{v-1}(2x_1^v+a_1+ \qquad (9)$$
$$+2^{s-v+1}c+2^{s-v-p+2}t,2x_2^v+a_2+2^{s-v+1}c,$$
$$\dots,2x_n^v+a_n+2^{s-v+1}c+2^{s-v-p+2}t)\cdot$$
$$\cdot e^{\frac{2\pi i\left[(2x_1^v+a_1+2^{s-v+1}c+2^{s-v-p+2}t)\times\right.}{2^s}}$$
$$e^{\frac{\left.\times(y_1^v+2^{s-1}b_1+2^{s-v+1}c+2^{s-v-p+2}t)\right]}{2^s}}\cdot$$
$$\cdot e^{\frac{2\pi i\left[(2x_2^v+a_2+2^{s-v+1}c)(y_2^v+2^{s-1}b_2+2^{s-v+1}c)\right]}{2^s}}\cdot\dots\cdot$$
$$\cdot e^{\frac{2\pi i\left[(2x_n^v+a_n+2^{s-v+1}c)(y_n^v+2^{s-1}b_n+2^{s-v+1}c)\right]}{2^s}}$$

3 THE OBTAINED RESULTS

For the algorithm testing program in the programming language C++ has been written for two- and three-dimensional signal. The testing was conducted on PC with following characteristics:

Processor: AMD FX-4170 4.2 GHz;

RAM: 8 GB;

Operating system: Windows 7.

Was compared with a standard algorithm for the discrete Fourier transform in the environment of Matlab 7.5.0 (R2007b) in two- and three-dimensional case. Test results are shown in seconds in tables.

Table 1 shows a comparison runtime in seconds of the two-dimensional FFT by analogue Cooley-Tookey algorithm and a standard algorithm for computing two-dimensional FFT in Matlab.

Table 2 shows a comparison runtime in seconds of the three-dimensional FFT by analogue Cooley-Tookey algorithm and a standard algorithm for computing three-dimensional FFT in Matlab.

Table 3 shows a comparison runtime in seconds of the parallel version two-dimensional FFT by analogue Cooley-Tookey algorithm and parallel standard algorithm for computing two-dimensional FFT by combination one-dimensional FFT.

Table 1: Calculating 2D FFT.

Size signal	2D FFT Matlab	2D FFT Cooley-Tukey algorithm analog	Speedup C++
128*128	0.001	0.001	~1
256*256	0.005	0.004	~1
512*512	0.027	0.017	~1.6
1024*1024	0.125	0.087	~1.4
2048*2048	0.620	0.389	~1.6
4096*4096	2.634	1.637	~1.6
8192*8192	13.609	6.904	~2
16384*16384	-	20.383	

Table 2: Calculating 3D FFT.

Size signal	3D FFT Matlab	3D FFT Cooley-Tukey algorithm analog	Speedup C++
32*32*32	0.002	0.002	~1.0
64*64*64	0.028	0.020	~1.4
128*128*128	0.282	0.188	~1.5
256*256*256	2.546	1.660	~1.5
512*512*512	-	14.736	

Table 3: Parallel calculating 2D FFT.

Size signal	Number of processes	Combination 1D FFT	2D FFT Cooley-Tukey algorithm analog	Speedup Cooley-Tukey
1024*1024	1	0.112	0.057	~1.6
	2	0.142	0.070	~1.0
	4	0.154	0.099	~0.8
	8	0.257	0.092	~0.7
	16	0.330	0.088	~0.5
2048*2048	1	0.516	0.275	~1.7
	2	0.512	0.396	~1.2
	4	0.596	0.407	~1.1
	8	1.045	0.345	~0.9
	16	1.195	0.453	~0.8
4096*4096	1	2.193	1.355	~1.7
	2	2.399	1.194	~1.4
	4	2.393	2.098	~1.2
	8	4.412	1.946	~1.1
	16	3.946	1.912	~1.1
8192*8192	1	12.538	4.957	~1.7
	2	10.509	5.245	~1.4
	4	11.753	7.848	~1.2
	8	18.551	8.162	~1.1
	16	18.196	8.907	~1.2

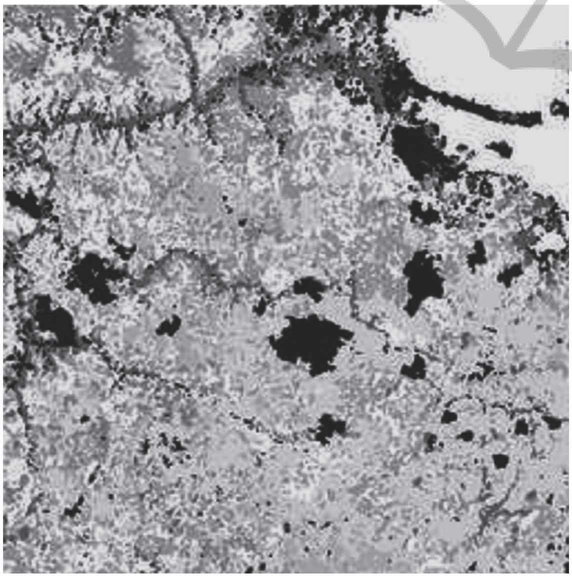

Figure 1: Example of two-dimensional signal.

4 CONCLUSIONS

The modified algorithm of the n-dimensional fast Fourier transform by analogue of the Cooley-Tukey algorithm requires $\frac{2^n - 1}{2^n} N^n \log_2 N$ complex operations of multiplications and $nN^n \log_2 N$ additions , where $N = 2^s$ is number of counts in the one of the coordinates (Starovoitov, 2010). Standard algorithm requires $nN^n \log_2 N$ complex multiplications and $nN^n \log_2 N$ complex additions. The modified algorithm requires less complex than the standard method, and runs 1.5 times faster than analogue in Matlab.

ACKNOWLEDGEMENTS

Work performed under the state order of the Ministry of Education and Science if the Russian Federation in the Siberian Federal University to perform R&D in 2014 (Task No 1.1462.2014/K). Project title: "Algebraic and analytic methods for creating algorithms for solving differential and polynomial systems: factorization, resolution of singularities and the optimal lattice"

REFERENCES

Dudgeon, D. E. and Mersereau, R. M., 1983. *Multidimensional Digital Signal Processing,* Prentice Hall.

Blahut, R. E., 1985. *Fast Algorithms for Digital Signal Processing*, Addison-Wesley Press.

Tutatchikov V. S., Kiselev O. I., Noskov M. V., 2013. "Calculating the n-Dimensional Fast Fourier Transform", *Pattern Recognition and Image Analysis,* vol. 23, no. 3, pp. 429-433.

Gonzalez, R. C., Woods, R. E., Eddins, S. L., 2009. *Digital Image Processing Using MATLAB*, Gatesmark Publishing. Knoxville.

Starovoitov, A. V., 2010. "On multidimensional analog of Cooley-Tukey algorithm", *Reporter Siberian State Aerospace University named after academician M.F.Reshetnev*, no. 1 (27), pp. 69-73.