Implementing FFT Based Cosine Transforms

Dániza C. Morales Berrios
Advisor: Dr. Jaime Seguel
Electrical and Computer Department
University of Puerto Rico – Mayagüez

daniza@cs.uprm.edu

Abstract

The Cosine Transform (CT) has several applications in science and engineering. Among them are numerical solutions for boundary value problems, and image processing, Fast algorithms for computing the cosine transform based on the Cooley – Tukey fast Fourier transform (FFT) have been proposed.
This article provides a mathematical framework for Swarztrauber's fast cosine transform, and establishes the mathematical basis for their generalization to a family of FFT based trigonometric transform. Some considerations about automatic code generation are also made.

1. Introduction

The FFT is a very well known method for the efficient computation of the discrete transform (DFT) of an N-point sample $(x_l : \ l = 0,...,N-1)$:

$$\hat{x}_k = \frac{1}{N}\sum_{l=0}^{N-1} x_l w \qquad (1).$$

where

$$w_N = e^{\frac{-2pi}{N}}$$

If the N-point sample is even, this is:

$$x_l = x_{n-l} \qquad l = 0,1,...,N-1 \qquad (2).$$

Then the DFT is also even. Furthermore, using (2), the following straight relation between DFT and CT can be established.

$$\hat{x}_k = x_0 + \sum_{l=0}^{\frac{N}{2}-1} x_l \left(e^{\frac{-2pkli}{N}} + e^{\frac{2pkli}{N}} \right) + (-1)^k x_{\frac{N}{2}}$$

$$= x_0 + 2\sum_{l=0}^{\frac{N}{2}-1} x_l \cos\left(\frac{2pkl}{N}\right) + (-1)^k x_{\frac{N}{2}}$$

$$k = 1,..,\frac{N}{2}$$

This relation suggests the use of the FFT for computing the CT. A pioneering work in that direction was done by Swarztrauber [6], who observed that the relation between the DFT and the CT could be carried over to each one of the intermediate computations of the FFT, generating that a truly fast algorithm for the cosine transform. In Swarztrauber's article the algorithm is expressed in terms of diagrams and no mathematical foundation is provided. This is a limitation both, for implementers and theoreticians who may wish to extend these ideas to more general trigonometric transforms.

This article provides the essence of a mathematical framework for Swarztrauber's fast cosine transform, and establishes the mathematical basis for their generalization to a whole family of FFT based trigonometric transform. This generalization allows us to propose a meta-algorithm for generating fast trigonometric transform's code automatically.

2. Mathematical Foundations

In this section the mathematical features of the Cooley-Tukey FFT are outlined in a language suitable for the application of symmetries of the form of:

$$x[l] = \pm x[ml] \qquad (4)$$

where m and N are relatively prime and all index operations are performed modulo N. For simplicity we assume $N=2^r$ but all results apply with minor modifications to any composite N.

For a fix $s, 1 \le s \le r-1$, and $v = 0,\ldots, 2^s-1$ we define:

$$\hat{x}_v[k] = \sum_{l=0}^{2^s-1} x[2^{r-s} l + v] w_{2^s}^{kl} \qquad (5)$$

where $k = 0,\ldots,2^{r-s}-1$

The following relation holds for s=1,..r-1.

$$\hat{x}_{\langle v \rangle_{2^{r-s-1}}}[k + d2^s] = \sum_{d=0}^{1} (-1)^d w_{2^{s+1}}^{k} \hat{x}_v[k] \quad (6)$$

$k = 0,\ldots,2^{s-1}$

$v = 0,\ldots,2^{r-s}-1$

With this notation, the Cooley-Tukey FFT is expressed as follows:

Step 1:

o

- For each k=0,…, 2 -

The fast two steps of a 16 point FFT,

notation are:

Phase 1

$$\begin{cases} \hat{x}_0[0] = x_0 + x_8 \\ \hat{x}_0[1] = x_0 - x_8 \end{cases} \quad \begin{cases} \hat{x}_4[0] = x_4 + x_{12} \\ \hat{x}_4[1] = x_4 - x_{12} \end{cases} \quad \begin{cases} \hat{x}_2[0] = x_2 + x_{10} \\ \hat{x}_2[1] = x_2 - x_{10} \end{cases} \quad \begin{cases} \hat{x}_6[0] = x_6 + x_{14} \\ \hat{x}_6[1] = x_6 - x_{14} \end{cases}$$

$$\begin{cases} \hat{x}_1[0] = x_1 + x_9 \\ \hat{x}_1[1] = x_1 - x_9 \end{cases} \quad \begin{cases} \hat{x}_5[0] = x_5 + x_{13} \\ \hat{x}_5[1] = x_5 - x_{13} \end{cases} \quad \begin{cases} \hat{x}_3[0] = x_3 + x_{11} \\ \hat{x}_3[1] = x_3 - x_{11} \end{cases} \quad \begin{cases} \hat{x}_7[0] = x_7 + x_{15} \\ \hat{x}_7[1] = x_7 - x_{15} \end{cases}$$

$$\begin{cases} \hat{x}_0[0] = \hat{x}_0[0] + \hat{x}_4[0] \\ \hat{x}_0[1] = \hat{x}_0[1] + w_4 \hat{x}_4[1] \\ \hat{x}_0[2] = \hat{x}_0[0] - \hat{x}_4[0] \\ \hat{x}_0[3] = \hat{x}_0[1] - w_4 \hat{x}_4[1] \end{cases} \quad \begin{cases} \hat{x}_2[0] = \hat{x}_2[0] + \hat{x}_6[0] \\ \hat{x}_2[1] = \hat{x}_2[1] + w_4 \hat{x}_6[1] \\ \hat{x}_2[2] = \hat{x}_2[0] - \hat{x}_6[0] \\ \hat{x}_2[3] = \hat{x}_2[1] - w_4 \hat{x}_6[1] \end{cases}$$

$$\begin{cases} \hat{x}_1[0] = \hat{x}_2[0] + \hat{x}_5[0] \\ \hat{x}_1[1] = \hat{x}_2[1] + w_4 \hat{x}_5[1] \\ \hat{x}_1[2] = \hat{x}_2[0] - \hat{x}_5[0] \\ \hat{x}_1[3] = \hat{x}_2[1] - w_4 \hat{x}_5[1] \end{cases} \quad \begin{cases} \hat{x}_3[0] = \hat{x}_3[0] + \hat{x}_7[0] \\ \hat{x}_3[1] = \hat{x}_3[1] + w_4 \hat{x}_7[1] \\ \hat{x}_3[2] = \hat{x}_3[0] - \hat{x}_7[0] \\ \hat{x}_3[3] = \hat{x}_3[1] - w_4 \hat{x}_7[1] \end{cases}$$

Phase 3

$$\begin{cases} \hat{x}_0[0] = \hat{x}_0[0] + \hat{x}_2[0] \\ \hat{x}_0[1] = \hat{x}_0[1] + w_8 \hat{x}_2[1] \\ \hat{x}_0[2] = \hat{x}_0[2] + w_8^2 \hat{x}_2[2] \\ \hat{x}_0[3] = \hat{x}_0[3] + w_8^3 \hat{x}_2[3] \\ \hat{x}_0[4] = \hat{x}_0[0] - \hat{x}_2[0] \\ \hat{x}_0[5] = \hat{x}_0[1] - w_8 \hat{x}_2[1] \\ \hat{x}_0[6] = \hat{x}_0[2] - w_8^2 \hat{x}_2[2] \\ \hat{x}_0[7] = \hat{x}_0[3] - w_8^3 \hat{x}_2[3] \end{cases} \quad \begin{cases} \hat{x}_1[0] = \hat{x}_1[0] + \hat{x}_3[0] \\ \hat{x}_1[1] = \hat{x}_1[1] + w_8 \hat{x}_3[1] \\ \hat{x}_1[2] = \hat{x}_1[2] + w_8^2 \hat{x}_3[2] \\ \hat{x}_1[3] = \hat{x}_1[3] + w_8^3 \hat{x}_3[3] \\ \hat{x}_1[4] = \hat{x}_1[0] - \hat{x}_3[0] \\ \hat{x}_1[5] = \hat{x}_1[1] - w_8 \hat{x}_3[1] \\ \hat{x}_1[6] = \hat{x}_1[2] - w_8^2 \hat{x}_3[2] \\ \hat{x}_1[7] = \hat{x}_1[3] - w_8^3 \hat{x}_3[3] \end{cases}$$

Phase 4

$$\begin{cases} \hat{x}[0] = \hat{x}_0[0] + \hat{x}_1[0] \\ \hat{x}[1] = \hat{x}_0[1] + w_{16} \hat{x}_1[1] \\ \hat{x}[2] = \hat{x}_0[2] + w_{16}^2 \hat{x}_1[2] \\ \hat{x}[3] = \hat{x}_0[3] + w_{16}^3 \hat{x}_1[3] \\ \hat{x}[4] = \hat{x}_0[4] + w_{16}^4 \hat{x}_1[4] \\ \hat{x}[5] = \hat{x}_0[5] + w_{16}^5 \hat{x}_1[5] \\ \hat{x}[6] = \hat{x}_0[6] + w_{16}^6 \hat{x}_1[6] \\ \hat{x}[7] = \hat{x}_0[7] + w_{16}^7 \hat{x}_1[7] \\ \hat{x}[8] = \hat{x}_0[0] - \hat{x}_1[0] \\ \hat{x}[9] = \hat{x}_0[1] - w_{16} \hat{x}_1[1] \\ \hat{x}[10] = \hat{x}_0[2] - w_{16}^2 \hat{x}_1[2] \\ \hat{x}[11] = \hat{x}_0[3] - w_{16}^3 \hat{x}_1[3] \\ \hat{x}[12] = \hat{x}_0[4] - w_{16}^4 \hat{x}_1[4] \\ \hat{x}[13] = \hat{x}_0[5] - w_{16}^5 \hat{x}_1[5] \\ \hat{x}[14] = \hat{x}_0[6] - w_{16}^6 \hat{x}_1[6] \\ \hat{x}[15] = \hat{x}_0[7] - w_{16}^7 \hat{x}_1[7] \end{cases}$$

If the sample is even then:
For s=1,…,r-1

$$\tilde{v} = -v \bmod 2^{r-s}$$

$$\hat{x}_v[k] = w_{2^s}^{kc} x_{\tilde{v}}[-k] \qquad (7)$$

$$k = 0,...,2^s - 1$$

where c and $\tilde{v}$ are the integers that satisfy :

$$\langle -v \rangle_{2^r} = 2^{r-s} c + \tilde{v}$$

where : $\langle m \rangle_n = m \bmod n$

Formula (7) is crucial for reducing intermediate redundant computations. For example, the value of $\hat{x}_6[0]$ can be computed from $\hat{x}_2[1]$ using the facts that the blocks 6 and 2 are symmetric, and formula 7, as follows:

$$\hat{x}_6[0] = w_4 \hat{x}_2[1]$$

By using this type of replacements, all computations can be performed with only fundamental data. This fundamental data in turn, result from restricting the original data to a fundamental indexing set.
Example: a 16-point data sample is indexed by the set {0,1,2,3,…,15}. Since the datum x[n]=x[-n], the fundamental (non-redundant) data is indexed by {0,1,2,3,…,8}. Therefore, the fundamental indexing set is, in this case, about one half the original set.

The next example shows the data flow of the 16-point even FFT, after all the eliminations and replacements have been performed.

Phase 1

$$\begin{cases} \hat{x}_0[0] = x_0 + x_8 \\ \hat{x}_0[1] = x_0 - x_8 \end{cases} \quad \begin{cases} \hat{x}_4[0] = x_4 + x_{12} = 2x_4 \\ \hat{x}_4[1] = x_4 - x_{12} = 0 \end{cases} \quad \begin{cases} \hat{x}_2[0] = x_2 + x_{10} \\ \hat{x}_2[1] = x_2 - x_{10} \end{cases} \quad \begin{cases} \hat{x}_1[0] = x_1 + x_9 \\ \hat{x}_1[1] = x_1 - x_9 \end{cases} \quad \begin{cases} \hat{x}_3[0] = x_3 + x_{11} \\ \hat{x}_3[1] = x_3 - x_{11} \end{cases}$$

Phase 2

$$\begin{cases} \hat{x}_0[0] = \hat{x}_0[0] + \hat{x}_4[0] \\ \hat{x}_0[1] = \hat{x}_0[1] + w_4 \hat{x}_4[1] \\ \hat{x}_0[2] = \hat{x}_0[0] - \hat{x}_4[0] \end{cases} \quad \begin{cases} \hat{x}_2[0] = \hat{x}_2[0] + w_4 \hat{x}_2[1] \\ \hat{x}_2[1] = \hat{x}_2[1] + w_4 \hat{x}_2[0] \\ \hat{x}_2[2] = \hat{x}_2[2] - w_4 \hat{x}_2[1] \end{cases} \quad \begin{cases} \hat{x}_1[0] = \hat{x}_2[0] + w_4 \hat{x}_3[1] \\ \hat{x}_1[1] = \hat{x}_2[1] + w_4 \hat{x}_3[0] \\ \hat{x}_1[2] = \hat{x}_2[0] - w_4 \hat{x}_3[1] \end{cases}$$

Phase 3

$$\begin{cases} \hat{x}_0[0] = \hat{x}_0[0] + \hat{x}_2[0] \\ \hat{x}_0[1] = \hat{x}_0[1] + w_8 \hat{x}_2[1] \\ \hat{x}_0[2] = \hat{x}_0[2] + w_8^2 \hat{x}_2[2] \\ \hat{x}_0[3] = \hat{x}_0[3] + w_8^3 \hat{x}_2[1] \\ \hat{x}_0[4] = \hat{x}_0[0] - \hat{x}_2[0] \end{cases} \quad \begin{cases} \hat{x}_1[0] = \hat{x}_1[0] + \hat{x}_1[0] \\ \hat{x}_1[1] = \hat{x}_1[1] + w_8 \hat{x}_1[1] \\ \hat{x}_1[2] = \hat{x}_1[2] + w_8^2 w_4{}^2 \hat{x}_1[2] \\ \hat{x}_1[3] = \hat{x}_1[3] + w_8^3 w_4 \hat{x}_1[1] \\ \hat{x}_1[4] = \hat{x}_1[0] - w_8 \hat{x}_1[1] \end{cases}$$

Phase 4

$$\begin{cases} \hat{x}[0] = \hat{x}_0[0] + \hat{x}_1[0] \\ \hat{x}[1] = \hat{x}_0[1] + w_{16} \hat{x}_1[1] \\ \hat{x}[2] = \hat{x}_0[2] + w_{16}^2 \hat{x}_1[2] \\ \hat{x}[3] = \hat{x}_0[3] + w_{16}^3 \hat{x}_1[3] \\ \hat{x}[4] = \hat{x}_0[4] + w_{16}^4 \hat{x}_1[4] \\ \hat{x}[5] = \hat{x}_0[3] + w_{16}^5 \hat{x}_1[3] \\ \hat{x}[6] = \hat{x}_0[2] + w_{16}^6 \hat{x}_1[2] \\ \hat{x}[7] = \hat{x}_0[1] + w_{16}^7 \hat{x}_1[1] \\ \hat{x}[8] = \hat{x}_0[0] - \hat{x}_1[0] \end{cases}$$

3. Code Generating Algorithm

The elimination of redundancies involves several operations. In order to have an efficient algorithm, all these operations must be raised to the level of a meta algorithm which generates the actual executable code. Experiments have been performed using Matlab for the meta-algorithm, and generating the actual code in C.

The main steps of the code-generating algorithm are the following:

1. Compute a fundamental input set
   Compute the fundamental symmetries base transforms.
2. For each s=1,…,r-1
   Compute a fundamental set of blocks

Compute a fundamental output set for each block

Use the equation (6) to fill un-computed fundamental values.

4. Conclusions

A mathematical framework for the generation of highly efficient cosine transforms has been proposed. The framework can be used to generate code from user's friendly environments such as Matlab. The underlying mathematics remain almost the same where most complex symmetry relations, such those of the form of $x_l = \pm x_{ml+b}$ are considered. All these ideas extend naturally to composite transform length.

References:

[1] N. Ahmed, T. Natarajan, and K.R. Rao, *Discrete cosine transforms*, IEEE Trans. Comput., C-2 (1974), pp. 90-93.

[2] W.L. Brigs, *Further symmetries of in-place FFTs*. SIAM J. SCi. Stat.Comput.,3(1987), pp. 664-665.

[3] J. Cooley and J. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297-301

[4] J. Tukey, P. Lewis and P. Welch, *The fast Fourier transform algorithm: Programming considerations in the calculation of sine, cosine and Laplace transforms*, J/ Sound Vibrations, 12, (1970), pp. 315-337.

[5] J. Dollimore, *Some algorithms for use with the fast Fourier transform*. J. Inst. Math. Appl., 12(1973), pp. 115-117.

[6] W.D. Gentleman, *Implementating Chenshaw-Curtis quadrature II: Computing the cosine transform*, Comm. ACM, 15 (1972), pp. 343-346.

[7] S. Martucci, *Symmetric convolution and the discrete sine and cosine transforms*, IEE Trans. Signal Processing, 42 (1994), pp. 1038-1051.

[8] K. R. Rao and P. Yip, *Discrete Cosine Transforms*, Academic Press, New York, 1990.

[9] P.N. Swarztrauber, *Symmetric FFTs*, Math. Comp., 47, (1986), pp. 323-346.

[10] Z.wang and B.hunt , *The discrete W-transform*. Appl. Math.Comput.,16 (1985), pp.19-48.