TRADE-OFF'S IN THE COMPUTATION OF MONO- AND MULTI-DIMENSIONAL DCT'S

Martin VETTERLI*, Pierre DUHAMEL** and Christine GUILLEMOT***

* Dept. of EE and CTR, Columbia University, New York, NY 10027
** CNET/CRPE, 38 Av. du Gen. Leclerc, F-92 131 Issy Les Moulineaux, France
*** CCETT, Rue du Clos Courtel, BP 59, F-35 510 Cesson Sevigné, France

Abstract: The computation of the discrete cosine transform (DCT) is a computationally intensive problem in a number of signal processing applications, for example in image and video coding (where it is part of standard methods). This paper reviews possible trade-off's in the computation of the DCT. Lower bounds on the number of multiplications in the one dimensional case as well as known numbers in the 2 dimensional case are reviewed. Alternative structures for the computation of the 2D DCT are indicated. Improvements through scaling is discussed and implementation issues (both in hardware and software) are addressed.

I Introduction

The DCT [1] has by now become the standard decorrelation transform for the compression of 1D and 2D signals, and has also been proposed in the 3D case. Numerous fast algorithms have been proposed for its computation, and these algorithms can be divided into 3 classes:

i) Direct algorithms based on the factorization of the DCT matrix [3,8,7].
ii) Indirect algorithms using the DFT, permutations [11] and auxiliary operations [13].
iii) Optimal algorithms (in terms of number of multiplications) [4,6]. The optimal algorithms are either indirect (N odd) or direct (N a power of 2) and rely on results for the complexity of the DFT and of polynomial multiplication.

Algorithms for the 2D DCT have been proposed as well, and are usually indirect, that is permutations followed by a 2D DFT and some output rotations [12,14]. Best performance is obtained when true 2D FFT algorithms are used for the DFT part (like vector-radix or polynomial transform algorithms).

In coding applications, the output of the DCT can usually be scaled since it will be followed by a quantizer that can take this scaling into account. This can lead to reduction in computational complexity [9].

Note that the DCT is not self-inverse, and that inverse algorithms, especially in the multi-dimensional case, can be involved to derive. However, the transposition principle (see [15]) can be used to derive an inverse algorithm with the same additive and multiplicative complexity.

Finally, most results presented will be biased towards minimizing the number of real multiplications (and then possibly the number of additions as well). However, some processors (e.g. CORDIC's) use rotations as basic operations, and therefore, the number of rotations has to be minimized [9]. Only partial results are available on this subject which turns out to be harder to study due to the weaker underlying mathematical structure.

In the examples, emphasis will be put on transforms of small length (typically, $N = 8$), since these are of great practical importance in applications like image coding. Note that we use the short hands μ and α for multiplication and addition respectively.

II One dimensional DCT algorithms

II.1 Definitions and preliminary remarks

The length-N 1D forward and inverse DCT's are defined as:

$$X(k) = c(k) \sum_{n=0}^{N-1} x(n) cos(\frac{2\pi(2n+1)k}{4N}) \qquad (1a)$$

$$x(n) = \sum_{k=0}^{N-1} c(k) X(k) cos(\frac{2\pi(2n+1)k}{4N}) \qquad (1b)$$

$$c(0) = \sqrt{1/N}, \quad c(k) = \sqrt{2/N}, \quad k \neq 0$$

Note the symmetric definition of the weights between forward and inverse transform. In matrix notation, (1) becomes:

$$\mathbf{X} = \mathbf{T}\mathbf{x}\ , \qquad \mathbf{x} = \mathbf{T}^T\mathbf{X} \qquad (2)$$

since $\mathbf{T}$ is unitary ($\mathbf{T}^{-1} = \mathbf{T}^T$). Separating the weights $c(k)$ from the actual trigonometric transform leads to:

$$\mathbf{T} = \mathbf{C}\mathbf{T}' \qquad (3a)$$

$$\mathbf{C} = Diag[\sqrt{1/N}, \sqrt{2/N}, \ . \ . \ , \sqrt{2/N}] \qquad (3b)$$

From (2) and (3) it follows that:

$$(\mathbf{T}')^T \cdot \mathbf{C}^2 \cdot \mathbf{T}' = \mathbf{I} \qquad (4a)$$

$$\mathbf{C}^2 = Diag[1/N, 2/N, \ . \ . \ , 2/N] \qquad (4b)$$

Therefore, instead of distributing the weighting over both the forward and inverse transform, the normalization is usually done in a single step with multiplication by $\mathbf{C}^2$ (this is especially practical when N is a power of 2 and this reduces to shifts only). We will call $\mathbf{T}'$ the "denormalized DCT" ($X(0)$ has a norm which is bigger by $\sqrt{2}$ compared to the other coefficients). This transform is the one usually considered [13,6] because of its intimate relationship to the DFT.

When N is even, $X(N/2)$ involves multiplication by $cos((2n+1)\pi/4) = \pm\sqrt{1/2}$ which can be merged between forward and inverse transform. A "scaled DCT" can then be obtained with $c(0) = 1$ and $c(k) = \sqrt{2}, k \neq 0$. This will save 1 multiplication over the denormalized DCT at $k = N/2$ [9]. Note that in this case all the DCT vectors have the same norm (equal to $\sqrt{N}$).

II.2 Algorithms for the 1D DCT

a) Case N odd

Heideman has shown [6] that this computational problem is equivalent to a DFT on a real sequence of length N, and this with permutations and sign changes only (that is, no arithmetic cost). Thus:

CH2673-2/89/0000-0999 $1.00 © 1989 IEEE

$$\mu[DCT(2K+1)] = \mu[DFT(2K+1)] \quad (5a)$$

$$\alpha[DCT(2K+1)] = \alpha[DFT(2K+1)] \quad (5b)$$

where $\mu[.]$ and $\alpha[.]$ stands respectively for multiplicative and additive complexity of the problem [.]. Therefore, one can use optimal DFT algorithms [2,12] and for example, DCT(9) uses 10 multiplications, 1 trivial multiplication and 34 additions.

b) Case N even

It can be shown that the denormalized DCT is a part of the DFT of size $4N$ [13], and in such a way that multiplicative complexities can be added up [6]. This leads to the general complexity formula:

$$\mu DCT(N) = \frac{\mu DFT(4N) - \mu DFT(2N)}{2} \quad (7)$$

Note that (5a) is a particular case of (7). Now, the (multiplicative) complexity of the 1D DFT is well studied and thus, (7) provides the minimum number of multiplications for a DCT of arbitrary length. For example, if $N = 2^m$, then the number of multiplications is [6]:

$$\mu[DCT(2^m)] = 2^{m+1} - m - 2 \quad (8)$$

This result was derived independently by Duhamel as well [4] based on the relation between the DCT and a circular convolution of the same length. However, these optimal algorithms in the case 2^m lead to unreasonable number of additions. For example, an optimal algorithm for the length-8 DCT uses 11 multiplications but 58 additions (15 of which can be implemented as shifts) and its structure is not very regular (polynomial products modulo cyclotomic polynomials $U+1$, U^2+1 and U^4+1).

Therefore, more practical solutions have to be sought. The mapping of the DCT into a DFT of the same length [11] and some auxiliary operations leads to the following operation count (N even) [13]:

$$\mu[DCT(N)] = \mu[DFT(N)] + 3/2 \cdot N - 2 \quad (9a)$$

$$\alpha[DCT(N)] = \alpha[DFT(N)] + 3/2 \cdot N - 3 \quad (9b)$$

Note that the DFT is taken on real values, requiring thus half as many multiplications as in the complex case. For example, for $N = 8$ and with $DFT(8) = (2\mu, 20\alpha)$ one obtains $DCT(8) = (12\mu, 29\alpha)$, that is a total of 41 operations. Note that at the cost of one multiplication, this algorithm saves 14 additions over the optimal algorithm (neglecting the shifts). Note that the optimal algorithm will be in Winograd form while the sub-optimal one will not (even if the DFT is in Winograd form, the output rotations will follow after the output addition stage of the DFT). For $N = 8$, Heideman [6] proposed an algorithm in Winograd form (using a suboptimal product modulo (U^4+1)) that leads to $(13\mu, 32\alpha)$ or a total of 45 operations. Note that the Winograd form will become useful in multi-dimensional DCT's.

While direct schemes (factorization of the DCT matrix [8,7]) can achieve similar performance, they are usually more complex to derive and their numerical behavior is sometimes tricky [8]. Note that the algorithm by Chen et al. [3] uses 15μ and 26α (that is also 41 operations) for this small DCT, but becomes less competitive for larger DCT's.

II.3 Comments on scaling and inverse

The above discussion has focused on the denormalized DCT because of its direct relationship to the DFT. As mentioned before, a scaled DCT saves 1 multiplication over the above algorithms, and this for both the optimal and the sub-optimal algorithm. For $N = 8$, this leads to a 10μ and 11μ algorithm respectively, with unchanged number of additions. Now, the DCT is often followed by a quantizer. If the quantizer is implemented with a specific look-up table for each output (as might be the case in high speed hardware), then each output can have a different scaling factor (which is then corrected in the quantizer). This permits the denormalization of the rotations in the sub-optimal algorithm, saving another $N/2-1$ multiplications and additions. For $N = 8$, this leads to a $(8\mu, 26\alpha)$ algorithm (assuming the rescaling in the quantizers is free). In the optimal algorithm, such a denormalization of the outputs would save only 2 multiplications in the case $N = 8$, leading to 8 multiplications as well

For the inverse, we will use the transposition principle (see [15]) which guarantees equal complexity (both in terms of multiplications and additions) by flowgraph inversion. Note that proper scaling has to be done between the forward and inverse transform (all terms have to have the same scaling factor).

III Two dimensional DCT algorithms

Transforms of size N by N will be considered. Unlike in the 1D case, there are fewer results on the complexity of multi-dimensional transforms with common factors in the various dimensions. Some techniques are known (like the Winograd nesting and the polynomial transform) but the structure of the algorithms is often involved.

A two-dimensional DCT of size N by N is defined as:

$$X(k_1,k_2) = c(k_1)c(k_2) \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1,n_2) \cos\left(\frac{2\pi(2n_1+1)k_1}{4N}\right) \cos\left(\frac{2\pi(2n_2+1)k_2}{4N}\right) \quad (10)$$

$$c(0) = \sqrt{1/N}, \quad c(k) = \sqrt{2/N}, k \neq 0$$

Clearly, the transform is separable, and that is why row column computation is frequently used (requiring $2N$ DCT's of length N). Again, we will consider denormalized versions as well as scaled versions of the 2D DCT (which can be obtained from their 1D counterparts by applying the 1D version over rows and columns).

When N is odd, a two-dimensional version of Heideman's mapping [6] will transform the DCT into a same size DFT. In what follows we will however be concerned with the case when N is a power of 2, since this is the case in most applications.

In matrix notation, we can rewrite (10) using Kronecker products [2] as (where the two dimensional data arrays have been expanded into vectors by stacking columns):

$$\mathbf{X} = (\mathbf{T} \otimes \mathbf{T})\mathbf{x} = (\mathbf{C}\mathbf{T}' \otimes \mathbf{C}\mathbf{T}')\mathbf{x} \quad (11)$$

This can be rewritten, using properties of the Kronecker product [2], as:

$$\mathbf{X} = (\mathbf{C} \otimes \mathbf{C}) \cdot (\mathbf{T}' \otimes \mathbf{T}')\mathbf{x} \quad (12)$$

We will concentrate on the computation of the denormalized 2D DCT given by $\mathbf{T}' \otimes \mathbf{T}'$. First, if the 1D DCT algorithm is in Winograd form (see (6)) and uses m_0 non-trivial and m_1 trivial multiplications (this gives the size of the matrix $\mathbf{B}$), then we can write (similarly to (11) and (12)):

$$\mathbf{T}' \otimes \mathbf{T}' = (\mathbf{C} \otimes \mathbf{C}) \cdot (\mathbf{B} \otimes \mathbf{B}) \cdot (\mathbf{A} \otimes \mathbf{A}) \quad (13)$$

This is the so-called Winograd nesting and it leads to the following number of non-trivial multiplications:

$$\mu[DCT(N \times N)] = m_0^2 + 2m_0 m_1 \qquad (14)$$

For example, in the case $N = 8$ where $m_0 = 11$ and $m_1 = 1$ this leads to 143 multiplications (and a very large number of additions). This is to be compared with the row-column method which uses 16·11=176 multiplications. Now, consider the case when the 1D DCT is computed through a DFT followed by post-rotations:

$$\mathbf{T}' = \mathbf{R} \cdot \mathbf{F} \qquad (15)$$

Then, similarly to (11) and (12):

$$\mathbf{T}' \otimes \mathbf{T}' = (\mathbf{R} \otimes \mathbf{R}) \cdot (\mathbf{F} \otimes \mathbf{F}) \qquad (16)$$

that is, a 2D trigonometric transform followed by rotations. Both matrices $(\mathbf{R} \otimes \mathbf{R})$ and $(\mathbf{F} \otimes \mathbf{F})$ can now use Winograd's nesting as in (13). Consider for example the case $N = 8$. Since $\mathbf{R}$ uses 10 non-trivial and 1 trivial multiplication, and $\mathbf{F}$ uses 2 and 6, it follows that $(\mathbf{R} \otimes \mathbf{R})$ uses 120 multiplications and $(\mathbf{F} \otimes \mathbf{F})$ uses 28, that is a total of 148 multiplications. This is slightly more than the previous scheme, but with many less additions. The major point however in (16) is that a 2D DCT can be computed as a same size trigonometric transform and post-rotations. Now, the post-rotations consist of Kronecker products of elementary 2 by 2 rotations. In [15] it is shown that:

$$R_\alpha \otimes R_\beta = P^T \cdot \begin{pmatrix} R_{\alpha+\beta} & 0 \\ 0 & R_{\alpha-\beta} \end{pmatrix} P \qquad (17)$$

where P is a matrix of additions and subtractions only. From (17), it follows that 4 outputs of the 2D DCT can be obtained with 2 rotations or 1.5 μ per point. Therefore, taking simplifications into account, a 2D DCT can be obtained from a 2D DFT of the same size on real inputs at the cost of a permutation and:

$$\mu[ROT(N \times N)] = 3/2N^2 - 2N \qquad (18)$$

This algorithm was first described, although in a different form, in [14]. Note that the multiplicative complexity is dominated by the post-rotations when N is small (like 8 or 16) and that the optimization above is thus worthwhile.

Depending on the choice of the 2D DFT algorithm, several different 2D DCT algorithms are possible. A first one of interest is a row-column approach for the 2D DFT as follows. Perform real-DFT's on the columns of the data, then on the rows. Combine some of the outputs to obtain the true complex DFT, while taking hermitian symmetry into account. Then apply the rotations so as to obtain the 2D-DCT. This algorithm is some kind of intermediate between a pure row-column DCT and a full 2D treatment. For small size DCT's, its arithmetic complexity is low (112μ, 470α for an 8 by 8 DCT, and 672μ, 2662α for a 16 by 16 DCT), and its simple structure seems well suited for high throughput VLSI implementations.

A second approach is obtained by using a vector-radix-2 algorithm for the 2D FFT on real data [10]. Compared to the previous algorithm, this one results in a lower number of operations, especially for longer transforms (104μ, 462α for an 8 by 8 DCT, and 640μ, 2630α for a 16 by 16 DCT). Nevertheless, its implementation requires several type of butterflies, and this solution would thus be prefered in software (its structure is easely programmed).

Finally, one can consider a polynomial transform approach for the 2D FFT computation. The resulting algorithm has lower computational requirements (104μ, 462α for an 8 by 8 DCT, and 568μ, 2558α for a 16 by 16 DCT), but whether this reduction in arithmetic complexity translates into faster implementation remains to be shown, since for small sizes, the gain is small but the structure complex. Note that permitting denormalized rotations (assuming scaling is done in the quantizers) one can save additional multiplications in the final rotations. For example, 23 multiplications can be saved in the 8 by 8 DCT, leading to a 81μ algorithm.

The numbers above should be compared with the (192μ, 464α) of the row/column approach (that is, a total of 656 operations). Permitting denormalization, this last number of multiplications can be reduced to 128 by denormalizing the rotations in each set of DCT's (8μ each and a total of 16 DCT's).

Yet another alternative would be to use a polynomial transform approach directly on the 2D DCT. Although very involved, this approach seems very promising since numbers of multiplications as low as 56 can be obtained for an 8 by 8 DCT. Denormalizing some rotations is also feasible, and results in an algorithm requiring still fewer multiplications. This approach thus deserves further work and will be reported elsewhere.

IV Implementation issues

In specialized processors (like signal processors), it is often advantageous to write so-called "linear code" for a specific recurring application like the DCT. In that case, the running time will be dominated by the number of operations (unless very sophisticated pipelining have to be taken into account). In that sense, the algorithms described above should improve running speeds, especially if multiplication times are higher than addition times. Preliminary results [5] indicate the following running times for 8 by 8 DCT's on grey scale pictures of size 672 by 536 (on a general purpose computer):

- 9.8 sec. for the algorithm in [3]
- 8.0 sec. for the algorithm in [13] in a row/column fashion
- 4.4 sec. for the algorithm based on a row/column FFT followed by optimized rotations Note that denormalizing the rotation leads to a further gain of 7%.

In hardware (VLSI), algorithms based on vector-radix FFT's and a set of rotations are still relatively well-structured. For CORDIC type of processors, the output rotations are well-suited, but the 2D FFT needs some more investigation.

Note that most figures were given for the denormalized DCT. For example, an additional 3 multiplications can be saved for the scaled 2D DCT (at $k_1, k_2 = N/2$), but this gain is not significant unless one considers a scaled 1D DCT used in a row-column fashion which then saves 2N multiplications.

The issue of quantization error in the DCT/IDCT is of great concern in finite precision implementations. The only way to get a perfect DCT/IDCT pair with a finite number of bits (and perfect meaning no error at all) is to embed the DCT into a finite field [2,12] so that the inverse exists as well. This embedding costs however additional bits (corresponding to the maximal growth of the quantized transform, that is of the order the number of bits of the coefficients plus $Log_2 N$ bits due to N additions). Also, the "meaning" of the coefficients can get lost by this embedding (the notion of distance is not defined any longer as in the case of "real" coefficients). Note, however, that an application of the connection between length-2^n DCT and cyclic convolution [4] may solve this problem, by computing the cyclic convolution by means of a Number Theoretic Transform (NTT). The overall scheme is depicted in figure 1. Provided that the number of bits of the modulus is large enough, properties of NTT's will guarantee that the DCT coefficients will be exact (to

desired precision). Usual distance properties are then used to scale and quantize the transformed coefficients, which can be back transformed by the same means. All these properties will hold provided that the computations are performed with a modulus of at least:

$$m + n + \lceil Log_2[N] \rceil \tag{19}$$

where n is the number of bits of the input and m the number of bits used to code the transform coefficients. Transformed values could be truncated for coding purposes.

Note that reduction in computational complexity usually reduces the computation noise (by reducing the number of sources) at least as long as a well conditioned algorithm is used. However, the correlation between noise in different channels is increased in a fast DCT over the straight matrix multiplication. Thus, the issue of quantization needs more study, especially in the context of fast algorithms.

V Conclusion

An overview of some alternative algorithms for one and two dimensional DCT's has been given, together with discussion of scaling and implementation issues. Operation counts were derived for typical examples useful in image processing. It is possible to generalize the 2D schemes to 3D DCT's as well (see [15]). The result is that a 3D DCT can be obtained from a 3D DFT of same size on reals at the cost of permutations and $O[3/2N^3]$ multiplications. The scheme involves rotations on 8 output points at a time.

Acknowledgements: The first author wishes to thank A.Ligtenberg and C.Loeffler for helpful discussions on the issue of scaling, as well as M.T.Heideman for his communications, and acknowledges support from the American National Science Foundation under grants CDR-84-21402 and MIP-8808277.

References

[1] N.Ahmed, T.Natarajan, and K.R. Rao, "Discrete Cosine Transform", IEEE Trans. on Computers, Vol. C-23, pp.88-93, Jan. 1974.

[2] R.E.Blahut, **Fast algorithms for Digital Signal Processing**, Addison-Wesley, 1984.

[3] W-H.Chen, C.H.Smith, and S.C.Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform", IEEE Trans. on Communications, Vol. COM-25, pp.1004-1009, Sept. 1977.

[4] P.Duhamel and H.H'Mida "New 2^n DCT algorithms suitable for VLSI implementation," Proc. ICASSP-87, Dallas, April 1987, pp.1805-1808.

[5] Ch.Guillemot and P.Duhamel, "Structured and computationally efficient discrete cosine transform algorithms," submitted for pub.

[6] M.T.Heideman, "Computation of an odd-length DCT from a real-valued DFT of the same length," Submitted for pub., IEEE Trans. on Acoust., Speech, Signal Processing, 1988.

[7] H.S.Hou, "A fast recursive algorithm for computing the discrete cosine transform," IEEE Trans. on Acoust., Speech, Signal Processing, Vol. ASSP-35, No. 10, Oct. 1987, pp. 1455-1461.

[8] B.G.Lee, "A new algorithm to compute the discrete cosine transform," IEEE Trans. on Acoust., Speech, Signal Processing, Vol. ASSP-32, No. 6, Dec. 1984, pp.1243-1245.

[9] C.Loeffler, A.Ligtenberg and G.S.Moschytz, "Algorithm-architecture mapping for custom DSP chips ," Proc. Int. Symp. on Circ. and Systems, Helsinki, June 1988, pp.1953-1956.

[10] Z.J.Mou and P.Duhamel, "In-place, butterfly-style FFT of 2-D real sequences," submitted to IEEE T-ASSP, 1987.

[11] M.J.Narasimha, and A.M.Peterson, "On the Computation of the Discrete Cosine Transform," IEEE Trans. on Communications, Vol. COM-26, pp.934-936, June 1978.

[12] H.J.Nussbaumer, **Fast Fourier Transform and Convolution Algorithms**, Springer, Berlin, 1982.

[13] M.Vetterli and H.J.Nussbaumer, "Simple FFT and DCT Algorithms with Reduced Number of Operations," Signal Processing, Aug. 1984.

[14] M.Vetterli, "Fast 2-D Discrete Cosine Transform", Proc. of the 1985 IEEE Intl. Conf. on ASSP, Tampa, March 1985, pp. 1538-1541.

[15] M.Vetterli, "Trade-off's in the computation of mono- and multi-dimensional DCT's," Center for Telecom. Res. Tech., Rep. CU/CTR/TR-090-88-18, Columbia University, New York, 1988.

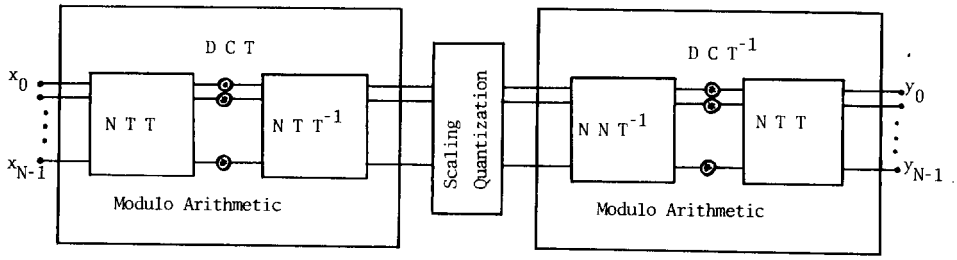

Figure 1: Computation of the DCT/IDCT in finite precision using Number Theoretic Transforms (NTT's). Note that only 8 multiplications per DCT are required.