DESIGN AND AUTOMATIC CODE GENERATION OF A TWO-DIMENSIONAL FAST COSINE TRANSFORM FOR SIMD DSP ARCHITECTURES

A. Lehmann, J. P. Robelly and G. Fettweis

Vodafone Chair for Mobile Communications, Technische Universität Dresden
01062 Dresden, Germany
lehmann@ifn.et.tu-dresden.de

ABSTRACT

Fast Algorithms for the computation of the two-dimensional Discrete Fourier Transform (DCT) can be described by means of elements of Multilinear Algebra. Multilinear Algebra offers not only a formalism for describing the algorithm, but it enables the derivation by pure algebraic manipulations of an algorithm that is well suited to be implemented in vector-SIMD signal processors with different levels of parallelism. The vector formulation of the two-dimensional DCT (2D-VDCT) can be implemented in a matrix oriented language and a suitable compiler generates code for the vector architecture. We show in this paper how important speedup factors can be achieved with this methodology.

1. INTRODUCTION

The two-dimensional DCT plays a paramount role in video and image compression techniques. Over the years many fast algorithms have been proposed for the computation of the DCT. An interesting work dealing with the derivation of fast algorithms for the computation of the DCT can be found in [1].

Most of the publications related to implementation issues of the DCT concentrate on VLSI implementations. We address in this paper the implementation of a fast algorithm for the DCT into SIMD-vector processors. In the last time, we have experienced how the SIMD-vector computational model has made its way from classical supercomputers to real-time embedded applications. In fact, vector signal processors have emerged upon the promise of delivering flexibility and processing power for computing number crunching algorithms at reasonable levels of power consumption. In [2], we have presented a novel micro-architecture for designing and implementing low-power, high-performance DSPs. Moreover, in [3] we presented a hardware design methodology that enables the rapid silicon implementation of SIMD-vector processors with different levels of parallelism.

The fast computation of signal transformations like the DCT is based on iterative divide-and-conquer algorithms: the transformation matrix is expressed as a function of smaller transformation matrices. Thus, the original computation that operates on vector spaces of a high dimensionality is reduced to the computation of smaller transformation matrices that operate on smaller vector spaces. The iterative formulation of the original transformation matrix is achieved by adequate permutation of the input samples. Elements of multilinear algebra are specially suitable for the description of this sort of algorithms. On the one hand, the rich framework offered by multilinear algebra allows for expressing the recursive nature of divide-and-conquer algorithms. On the other hand, it also enables the manipulation and derivation of new algorithms by exploiting pure algebraic properties.

Especially interesting are those algebraic manipulations that reveals the vector operations of the algorithm, since they lead to formulations of algorithms that process data in vector fashion. These ideas are discussed in detail in [4],[5], and they encouraged many researchers to publish a series of papers. Most of these papers address the derivation of vector algorithms for the classical example of the Fast Fourier Transform (FFT). Especially interesting is the work by Franchetti [6], where an algorithm for the vector computation of the FFT is presented.

In this paper we present the design of a vector algorithm for the two-dimensional DCT based on the framework of multilinear algebra. Once a suitable algorithm is designed, we implement it in a matrix oriented language like Matlab^{TM}. Such a language allows for expressing algorithms described in the notation of multilinear algebra. A suitable compiler can recognize these operators and generate a sequence of machine instructions. We show that important speedup factors are achieved by this methodology. The remainder of this paper is as follows. In section 2 we present some elements of multilinear algebra. In section 3 we use this algebraic framework for the derivation of the 2D-VDCT algorithm. In section 4 we introduce our compiler infrastructure and the results obtained from the automatic code generation. Finally, in Section 5 we present our conclusions.

2. THE ALGEBRAIC REPRESENTATION OF SIMD PARALLELISM

Let us assume the computation $\mathbf{y} = A\mathbf{x}$, where $\mathbf{y}, \mathbf{x}$ are some vectors whose components are scalars. The transformation matrix A defines an algorithm which in order to be computed in a serial computer requires a certain number of machine cycles c. Now, let us consider that the same transformation is to be computed in a vector processor with a level of parallelism v. It is a fact that in the same number of considered machine cycles c the transformation described by the matrix A can be computed for v different vectors. This can be expressed as

$$\tilde{\mathbf{y}} = (A \otimes I_v)\,\tilde{\mathbf{x}}, \tag{1}$$

where $\otimes$ is the Kronecker product, I_v is a $v \times v$ identity matrix, and $\tilde{\mathbf{x}}, \tilde{\mathbf{y}}$ can be regarded as vectors with v components and each component is itself a vector of the same dimensionality as $\mathbf{x}$ and $\mathbf{y}$ respectively. Thus, Equation (1) also captures the SIMD computational model for a parallelism of v. More generally, the expression

$$(I_b \otimes A \otimes I_v \otimes I_c) \tag{2}$$

$$P_K^{MLK} = \left(P_K^{MK} \otimes I_L\right)\left(I_M \otimes P_K^{LK}\right) \quad (7)$$

$$P_{ML}^{MLK} = \left(I_M \otimes P_L^{LK}\right)\left(P_M^{MK} \otimes I_L\right) \quad (8)$$

$$I_{ML} = I_M \otimes I_L \quad (9)$$

Table 1: Identities for manipulation of permutations and tensor products

can be efficiently implemented in a vector processor with a level of parallelism v, since Equation (1) is embodied here. As we can observe the Kronecker product, which is an operator from multilinear algebra, plays an important role in the description of algorithms for vector-SIMD processors. Consider the $m_1 \times n_1$ matrix A with entries $\left[a_{j,k}\right]$ for $j = 1,2,\ldots,m_1$ and $k = 1,2,\ldots,n_1$, and the $m_2 \times n_2$ matrix B, then the $m_1m_2 \times n_1n_2$ matrix C that results from the Kronecker product of A and B is defined as

$$C = A \otimes B = \begin{pmatrix} a_{1,1}B & a_{1,2}B & \ldots & a_{1,n_1}B \\ a_{2,1}B & a_{2,2}B & \ldots & a_{1,2}B \\ \vdots & \vdots & & \vdots \\ a_{m_1,1}B & a_{m_1,2}B & \ldots & a_{m_1,n_1}B \end{pmatrix}. \quad (3)$$

Another important concept from multilinear algebra is the direct sum. The direct sum of n arbitrary matrices is defined as

$$\bigoplus_{i=0}^{n-1} A_i = \begin{bmatrix} A_0 & 0 & \cdots & 0 \\ 0 & A_1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & A_{n-1} \end{bmatrix}. \quad (4)$$

To some extent, multilinear algebra can be regarded as the branch of algebra that deals with the construction of vector spaces of a higher dimensionality from a set of primitive vector spaces. The Kronecker product and the direct sum are the two fundamental concepts of algebra that enable this construction of vector spaces of a higher dimensionality [7].

Specially intriguing is the connection between Kronecker products and shuffle algebra. Davio, in his classical paper [8] established the connection between the Kronecker product of matrices and stride permutations. In his paper he proved the important so called commutation theorem of Kronecker products

$$(A \otimes B) = P_M^N (B \otimes A) P_L^N, \qquad N = ML, \quad (5)$$

where A, B are $M \times M$, $L \times L$ matrices and P_L^N is an N-point stride by L permutation. Additional useful identities for manipulating stride permutations are collected in table 1. Kronecker products present a series of other interesting algebraic properties. For example, for the above introduced square matrices A and B we can write [4]

$$A \otimes B = \left(A \otimes I_L\right)\left(I_M \otimes B\right). \quad (6)$$

3. 2D-VDCT

In this section we present the mathematical derivation of the two-dimensional DCT adapted to vector-SIMD processing. Firstly, we introduce the one-dimensional DCT algorithm and later we extend the algorithm to the two-dimensional case.

3.1 1-D Fast Cosine Transform Algorithm

In [9], Cvetković developed an algorithm for the one-dimensional DCT based on the algorithm derived by Hou [10]. The fast algorithm described in this paper presents a factorization of the N-point Type-II 1D-DCT as a product of sparse matrices. This factorization can be expressed in terms of elements of multilinear algebra as follows

$$\begin{aligned} \mathrm{DCT}_N &= Q_N \left[\prod_{k=0}^{q-2} \left(I_{2^k} \otimes A_{\frac{N}{2^k}}\right)\right] \left[\prod_{k=1}^{q} \left(I_{\frac{N}{2^k}} \otimes B_{2^k}\right)\right. \\ &\quad \left. \cdot \left(I_{\frac{N}{2^k}} \otimes \mathrm{DFT}_2 \otimes I_{2^{k-1}}\right)\right] R_N, \end{aligned} \quad (10)$$

where $q = \log_2 N$ and $\prod_{k=0}^{q} A_k = A_0 \cdot \ldots \cdot A_q$. The bit-reversal matrix can be defined in terms of stride permutations as

$$Q_N = \prod_{k=0}^{q-2} \left(I_{2^k} \otimes P_{2^{q-k-1}}^{2^{q-k}}\right).$$

A_N is a sparse matrix involved in the computation and is defined as

$$A_N = \left(I_{\frac{N}{2}} \oplus K_{\frac{N}{2}}\right), \quad (11)$$

where I_N is an $N \times N$ identity matrix and $K_N = Q_N L_N Q_N$ for the $N \times N$ matrix

$$L_N = \begin{bmatrix} 1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 1 & \ddots & \vdots \\ 0 & 0 & 1 & \ddots & 0 \\ \vdots & & \ddots & \ddots & 1 \\ 0 & & \cdots & 0 & 1 \end{bmatrix}. \quad (12)$$

The other sparse matrix involved in the computation is B_N, which is defined as

$$B_N = \left(I_{\frac{N}{2}} \oplus C_{\frac{N}{2}}\right), \quad (13)$$

where

$$C_{\frac{N}{2}} = \mathrm{diag}\left[\frac{1}{2\cos(\phi_m)}\right], \quad m = 0,1,\ldots,N/2-1 \quad (14)$$

and

$$\phi_m = \frac{2\pi(m+1/4)}{N}.$$

The matrix DFT_2 denotes the 2-point Discrete Fourier Transform (DFT) matrix and it is given by

$$\mathrm{DFT}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (15)$$

Finally, the input permutation matrix R_N is defined as

$$R_N = \left(I_{\frac{N}{2}} \oplus \overline{I}_{\frac{N}{2}}\right) P_2^N,$$

where $\overline{I}_{N/2}$ is the mirrored $N/2 \times N/2$ identity matrix. For example, $\overline{I}_{\frac{N}{2}}$ for $N = 8$ is given by

$$\overline{I}_{\frac{8}{2}} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

3.2 Derivation of the 2-D VDCT Algorithm

Pratt [11] points out that separable bilinear transformations can be expressed as two linear transformations: one operating over the rows and one operating over the columns. Since the 2D-DCT can be regarded as such a bilinear transformation we can write for the transformation of an $N \times N$ matrix X the following

$$(\mathrm{DCT}_N)\, X\, (\mathrm{DCT}_N)^T. \tag{16}$$

The computation of the 2D-DCT in the matrix space as in equation (16) is isomorph to a computation of the algorithm that operates onto a vector space. The vector space is constructed by stacking the rows of the matrix X. For this case the computation of the 2D-DCT becomes

$$(\mathrm{DCT}_N \otimes \mathrm{DCT}_N) \cdot \mathrm{Vec}\,(X), \tag{17}$$

where $\mathrm{Vec}\,(\cdot)$ represents the vectorization of a matrix in row-major order. Thus, the two-dimensional algorithm can be derived from the one-dimensional matrix form using the identity

$$\mathrm{DCT}_{N\times N} = \mathrm{DCT}_N \otimes \mathrm{DCT}_N. \tag{18}$$

This 2-D transformation matrix has to be applied to a vector $\mathbf{x}$ of N^2 elements. The vector $\mathbf{x}$ is obtained from the row-major ordering of the $N \times N$ input data array $X(n,n)$ and thus it is defined as

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_0^T \\ \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_{N-1}^T \end{bmatrix} \qquad \text{for} \quad \mathbf{x}_n = [X(n,1)\ldots,X(n,N-1)].$$

Using (6), we can write for equation (18) the following

$$\mathrm{DCT}_{N\times N} = (\mathrm{DCT}_N \otimes I_N)(I_N \otimes \mathrm{DCT}_N). \tag{19}$$

In order to derive an algorithm that process data in vector fashion, expressions of the form given by (1) are required. By using identity (5), (19) can be expressed as

$$DCT_{N\times N} = (DCT_N \otimes I_N) P_N^{N\cdot N} (DCT_N \otimes I_N) P_N^{N\cdot N}, \tag{20}$$

where the maximal SIMD parallelism of $v_{max} = N$ is obtained. The expression $P_N^{N\cdot N}$ denotes a transposition matrix that cannot be efficiently mapped to SIMD processors, since it does not match the form given by (2). For this reason, a further decomposition is required to obtain a simpler structure for vector processors. Using identities (7) and (8) in table 1 yields to a formulation adapted to a level of parallelism v. We obtain for the transposition

$$P_N^{N^2} = \left(I_{\frac{N}{v}} \otimes P_v^N \otimes I_v\right) \left(I_{\frac{N^2}{v^2}} \otimes P_v^{v^2}\right) \left(P_{\frac{N}{v}}^{\frac{N^2}{v}} \otimes I_v\right). \tag{21}$$

The term $(\mathrm{DCT}_N \otimes I_N)$ adapted to a SIMD vector length v, using the selected 1D-algorithm (10) and identity (9) is given by

$$\begin{aligned}(\mathrm{DCT}_N \otimes I_N) = &\prod_{k=0}^{q-2} \left(I_{2^k} \otimes P_{2^{q-k-1}}^{2^{q-k}} \otimes I_v \otimes I_{\frac{N}{v}}\right) \\ &\cdot \left[\prod_{k=0}^{q-2} (I_{2^k} \otimes A_{\frac{N}{2^k}} \otimes I_v \otimes I_{\frac{N}{v}})\right] \left[\prod_{k=1}^{q} \left(I_{\frac{N}{2^k}} \otimes B_{2^k} \otimes I_v \otimes I_{\frac{N}{v}}\right)\right. \\ &\cdot \left.\left(I_{\frac{N}{2^k}} \otimes \mathrm{DFT}_2 \otimes I_v \otimes I_{2^{(q-p+k-1)}}\right)\right] \left(R_N \otimes I_v \otimes I_{\frac{N}{v}}\right) \\ &q = \log_2 N,\ p = \log_2 v,\ p \le q. \end{aligned} \tag{22}$$

Applying (21) and (22) into (20), we obtain a fast 2-D VDCT algorithm for vector-SIMD processing. The derived algorithm is completely parameterizable by the transformation size N of the source data array and the available level of parallelism v of the used processor. Nearly all terms of the algorithm are vector computations matching expression (2) excepting for $\left(I_{N^2/v^2} \otimes P_v^{v^2}\right)$.

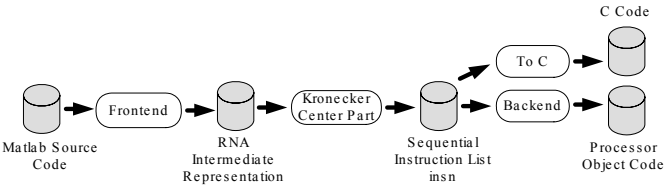

Figure 1: Block Diagram of the Compiler Infrastructure for Automatic Code Generation

4. IMPLEMENTATION AND RESULTS

The algorithm can be directly used for easy implementation in matrix-oriented languages like Matlab. In [3], we presented a compiler infrastructure for the automatic code generation for vector-SIMD processors. In figure 1, we can observe a block diagram of the compiler. The compiler stage known as Kronecker center part features the pattern matching of expressions embedded in the Matlab code, which have the form of equation (2). The Matlab code is programmed using functions that compute the different operators of multilinear algebra and stride permutations introduced in section 2. This compiler stage also generates a high level intermediate representation of vector instructions that will be processed by further stages of the compiler. This high level intermediate representation of the program resembles a linear instruction list.

The algorithm presented in section 3 was programmed in Matlab and object code was generated for our DSP processor cores with different levels of parallelism v. For N=16 speedup results for a raising number of data paths v are presented in figure 2. The speedup is computed taking as a reference the execution time required to compute the algorithm into a DSP processor with a level of parallelism $v = 4$. As we can observe from the diagram, we can expect to achieve a speedup factor of 1.76 if we implement the algorithm into a machine with $v = 8$ data paths. A more important gain on the speedup factor is achieved if the level of parallelism is $v = 16$. For this case some expensive permutations of (21) are cancelled out.

In this first implementation all Kronecker product terms

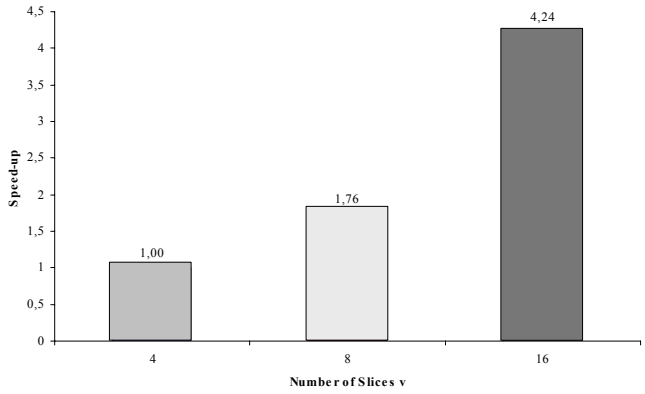

Figure 2: Speed-up factors for $\mathrm{DCT}_{16\times16}$

are computed separately. Improving performance is achievable by combining separate terms in (21) and (22). Thus, load/store-operations and cycles for data addressing are saved. For example, suitable terms for combined computation are

$$\left(I_{\frac{N}{2^k}} \otimes B_{2k} \otimes I_v \otimes I_{\frac{N}{v}}\right) \quad \text{and}$$
$$\left(I_{\frac{N}{2^k}} \otimes \mathrm{DFT}_2 \otimes I_v \otimes I_{2(q-p+k-1)}\right).$$

After merging this terms we obtain

$$\left(I_{\frac{N}{2^k}} \otimes \left(B_{2k}\left(\mathrm{DFT}_2 \otimes I_{2(k-1)}\right)\right) \otimes I_v \otimes I_{\frac{N}{v}}\right). \qquad (23)$$

In order to reduce the impact of the unique non-vector term of our algorithm, namely $P_v^{v^2}$, it is important to find an efficient implementation. A proposed approach [12] is described in figure 3 for $v = 4$. Our processor architecture is furnished with an interconnection network that supports the required data transfers between data paths in $i = \log_2 v$ steps.

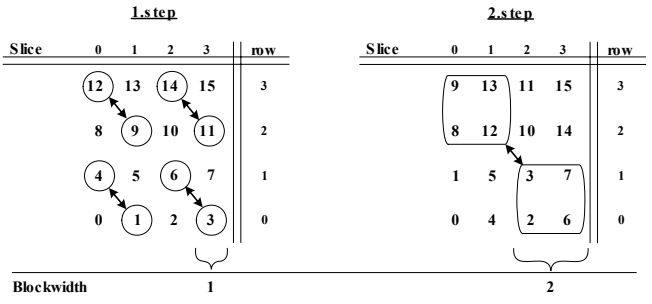

Figure 3: Data transfers for computing $P_v^{v^2}$ for $v = 4$

5. CONCLUSION

The 2D-VDCT algorithm has been derived by pure mathematical means using concepts of multilinear algebra. Since almost all terms involved in the computation of 2D-VDCT process data in vector fashion, the algorithm is specially efficient for vector-SIMD processors. The algorithm is completely parameterized for a certain transformation size N and the available SIMD level of parallelism v. We have presented a compiler infrastructure that can generate code for our family of DSP cores directly from a Matlab program. The computation of the algorithm in our processor with $v = 16$ data paths can be up to a factor of 4,24 times faster than the implementation of the algorithm into a processor with $v = 4$ data paths. We believe that the methodology presented in this paper offers an interesting approach to close the gap between fast algorithms for signal processing, compiler technology and processor architecture.

REFERENCES

[1] M. Püschel and J.M.F. Moura, "The Algebraic Approach to the Discrete Cosine and Sine Transforms and their Fast Algorithms," *SIAM Journal of Computing*, 2003.

[2] G. Cichon, J. P. Robelly, H. Seidel, E. Matus, M. Bronzel and G. Fettweis, "Synchronous Transfer Architecture (STA)," *Lecture Notes on Computer Science*, S.Vassiliadis, Ed. Berlin, Germany: Springer-Verlag, July 2004.

[3] J. P. Robelly, G. Cichon, H. Seidel, and G. Fettweis, "A hw/sw design methodology for embedded SIMD vector signal processors," *International Journal of Embedded Systems IJES*, January 2005.

[4] R. Tolimeri, M. An and C. Lu, *Algorithms for Discrete Fourier Transforms and Convolution*. New York, USA: 2nd edition, Springer-Verlag, 1997.

[5] R. Tolimeri, M. An and C. Lu, *Mathematics of Multidimensional Fourier Transform Algorithms*. New York, USA: 2nd edition, Springer-Verlag, 1997.

[6] F. Franchetti and M. Püschel, "Short Vector Code Generation for Discrete Fourier Transform," *Proc. International Parallel and Distributed Processing Symposium (IPDPS)*, 2003, pp. 58–67.

[7] H. Boseck, *Tensorräume*. Berlin: VEB Deutscher Verlag der Wissenschaft, 1972.

[8] M. Davio, "Kronecker Products and Shuffle Algebra," *IEEE Trans. on Computers*, vol. C-30, number 2,pp. 116-125, feb. 1981.

[9] Z. Cvetković and M. V. Popović, "New Fast Recursive Algorithms for the Computation of the Discrete Cosine and Sine Transforms," *IEEE Trans. on Signal Processing*, vol.40, pp.2083–2086, 1992.

[10] H. S. Hou, "A Fast Recursive Algorithm for Computing the Discrete Cosine Transform," *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. ASSP-35, no.10, pp. 1455–1461, 1987.

[11] W. Pratt, *Digital Image Processing*. New York, USA: John Wiley and Sons, 1991.

[12] J. C. Carlach, P. Penard, and J. L. Sicre, "TCAD: a 27 MHz 8x8 discrete cosine transform chip," in *Proc. IEEE ICASSP*, Glasgow, UK, May 23-26. 1989, pp. 2429-2432.