Parallelizing a Set of 2-D Frequency Transforms in a Flexible Manner

M. Fleury and A. F. Clark

Electronic Systems Engineering Department, University of Essex

Wivenhoe Park, Colchester, CO4 4SQ, U.K

tel: +44 - 1206 - 872795

fax: +44 - 1206 - 872900

e-mail fleum@essex.ac.uk

Abbreviated title: Parallelizing 2-D Frequency Transforms

Abstract

The implementation of parallel 2-D frequency transforms intended for the acceleration of image-processing algorithms is described. The way these routines fit into a wider generic format for such parallel routines is also indicated. The paper touches on the design decisions needed to marry choice of efficient routine with wide utility. Due consideration is given to auxiliary functions. Details of the book-keeping required to enable real-valued data to be efficiently transformed in a parallel setting are included.

1 Introduction

This paper describes the design and implementation of a generic suite of 2-D frequency transform programs intended for message-passing multicomputers. The objective was an image-processing library implemented in a common format which allows the user to select a serial or a parallel mode of operation [1]. The Fourier transform is central to the class of frequency

transforms. 2-D Discrete Fourier transform (DFT) algorithms include: the Vector Radix (VR) method [2]; the polynomial transform (PT) [3]; and the row-column (RC) algorithm [4]. A hybrid version of the RC algorithm [5] is appropriate to restricted memory or possibly shared-memory multiprocessors. The VR method, when parallelized [6], does not allow data distribution and collection to be overlapped with computation. The PT algorithm does not have a sufficiently regular structure which would present a problem when mapping the algorithm between different parallel machines. All algorithms have synchronization points. As with others concerned with generalized solutions on medium-grained machines [7] the RC algorithm was chosen. The degrees of freedom inherent in the RC algorithm allow adaptation to a variety of parallel environments and the algorithm can also be adapted for a 1-D transform.

Having developed a 2-D DFT, we went on to use the RC algorithm as a basis for a parallel Discrete Cosine Transform (DCT), with a saving in development time.

For large images, an alternative implementation of the 2-D DFT was preferred by means of a block method of data distribution. The techniques developed for real-valued images in the previous programs were extended when applied to a convolution program using the alternative method.

The algorithms have been implemented on a transputer-based machine, the Parsys Supernode, and were later easily ported to PVM [8], which is a *de facto* standard for a wide range of parallel platforms. For the Supernode implementation, a reusable generic hardware structure [9] was designed, which is shown in Figure 1 (though the apparatus for global communication was not needed for the frequency transforms). This structure is intended to reduce the scaling problem on store-and-forward networks at the same time utilizing the central processor more fully. Other low-level image-processing algorithms developed at the same time use the same common structure, e.g. [10] which does include global communication. The PVM version, when run on workstations linked by a LAN, simulates the design by what is in effect a master linked by a bus to a set of slaves. PVM implementations vary in the granularity of the tasks supported according to the network bandwidth, the efficiency of message buffering and the

overhead imposed by the dynamic model of parallelism. The porting problem is eased in that the number of message-passing modes is limited in PVM, implying that these modes will be efficiently implemented.

The frequency transforms are placed within a common framework, using data-farming, which is also suitable for the parallelisation of other low-level image-processing routines on general-purpose parallel environments. This is in contrast to work elsewhere on bespoke routines. The parallel algorithms build on existing optimized sequential algorithms, again reducing development time. A DCT and convolution algorithm are newly parallelised according to the generic method. The paper considers the important but sometimes ignored issue of choice of auxiliary routines, pointing out performance trade-offs. The processing of real-valued image data is focussed on, and the paper newly considers efficient sequencing of processing for a block-based convolution program.

Since processor clock speed is estimated to increase at an annual rate of 80% (though not processing speed due to the memory access problem [11]), any timings recorded should only be viewed for their relative and not absolute value. Modular parallel systems have recently been engaged to accelerate the well-known MatLab dsp computing environment, i.e. in the Paramex system [12] reducing the time for a wavelet transform from about 300s on a 486 PC at 66 MHz to 15s using 3L Parallel C [13] (also used for this study and suitable for SHARC processor networks) on five TMS320C40 at 40 MHz. A DFT-based parallel wavelet transform on 'C40s is described in [14] and DFTs on 'C40s in [15].

2 The Parallel 2-D FFT

2.1 The RC Algorithm

There has been substantial algorithmic development since the rediscovery of the binary-recursive Fast Fourier Transform (FFT) algorithm, including 1-D FFT parallel algorithms for vector [16], hypercube [17], and shared-memory machines [18]. On medium-grained machines a likely route is to perform a long 1-D FFT by means of a 2-D algorithm.

Define the 1-D transform by

$$X_k = \sum_{n=0}^{P-1} x_n e^{ink2\pi/P}, \tag{1}$$

for $k = 0, 1, \ldots P-1$ and $i = \sqrt{-1}$. If $P = P_0 P_1$, by means of the index mapping

$$n = s_0 + s_1 P_0, \quad k = m_1 + m_0 P_1, \tag{2}$$

the two-dimensional arrays (notice the indices' suffixes show the range of the indices)

$$\begin{aligned} x(s_0, s_1) &= x_n, \qquad s_0 = 0, 1, \ldots, P_0 - 1, \quad s_1 = 0, 1, \ldots, P_1 - 1 \\ X(m_1, m_0) &= X_k, \quad m_1 = 0, 1, \ldots, P_1 - 1, \quad m_0 = 0, 1, \ldots, P_0 - 1, \end{aligned} \tag{3}$$

can be employed to re-write (1) as

$$X(m_1, m_0) = \sum_{s_0=0}^{P_0-1} \omega_{P_0}^{s_0 m_0} \omega_P^{s_0 m_1} \sum_{s_1=0}^{P_1-1} x(s_0, s_1) \omega_{P_1}^{s_1 m_1} \tag{4}$$

to form P_0 transforms of length P_1:

$$X_1(s_0, m_1) = \omega_P^{s_0 m_1} \sum_{s_1=0}^{P_1-1} x(s_0, s_1) w_{P_1}^{s_1 m_1} \tag{5}$$

and P_1 transforms of length P_0:

$$X_2(m_0, m_1) = \sum_{s_0=0}^{P_0-1} X_1(s_0, m_1) \omega_{P_0}^{s_0 m_0}, \tag{6}$$

setting $\omega_n = e^{i2\pi/n}$ for some integer n, when the L.H.S. of (6) is in digit-reversed (here transposed) order.

The 2-D DFT of size $N \times N$ (with no loss of generality from using a square image) may be written as:

$$X_{jk} = \sum_{p=0}^{N-1} \sum_{q=0}^{N-1} x_{pq} e^{i(jp+kq)2\pi/N}, \tag{7}$$

for $j, k = 0, 1, \ldots N-1$, omitting normalization factors, and can be rewritten in separable form as

$$X_{jk} = \sum_{p=0}^{N-1} \left[\sum_{q=0}^{N-1} x_{pq} e^{i(jp)2\pi/N} \right] e^{i(kq)2\pi/N}. \tag{8}$$

Equation 8 is the RC algorithm, and it will be seen that it has the same structure as (5) except for a multiplicative factor (the 'twiddle' or rotation term).

As is well-known, the time complexity of a 1-D DFT is reduced from $O(N^2)$ to $O(N \log N)$ if an FFT is employed. By performing the term in parentheses in (8) first the complexity of the 2-D RC algorithm is $O(N^2 \log N)$, since there are $2N$ 1-D transforms in all.

2.2 Parallel Implementation

Many low-level image-processing algorithms, such as spatial filters, are completely localized in their data references. If adjacent image data are overlapped at boundaries then at a small additional cost a data-farming programming paradigm can be employed, in which the only communication is between worker process and data farmer. Provided that $n \gg p$, where n is the number of sub-images farmed out – for instance an image row, and p is the number of processors, then load-balancing can be made semi-automatic by having returning processed work form an implicit request for more data. Initiation of data-farming is by a static scheduling of work, e.g. a number of rows to each worker processor, which ideally should take no longer than the time to process and return one item of work. There is also a wind-down phase when outstanding work is collected [19].

The chief advantage of the RC algorithm for a parallel implementation is that data-farming of image rows/columns can be utilized to offset data-loading by computation provided thought is given to local buffering in order to offset communication latency. The number of rows/columns farmed out per work packet can be varied to suit the communicate/computate ratio of the machine. Notice that due to the linearity of the Fourier transform, the granularity is not restricted to the image size. An overlap-save method (Section 4.1) can be employed to further decompose each row/column. Global optimization techniques are an alternative way to obtain a balanced distribution of work but such approaches can require costly preliminary calculation though, however, they may be suitable for embedded applications. Where system fluctuation, congestion or communication overhead causes the time for any FFT to be non-deterministic then order statistics can predict the likely finishing time and similarly the optimum FFT size [20].

2.3 1-D FFT Algorithm Selection

DSP implementations of the FFT are distinguished by the required regularity of the algorithm, which usually means single-radix algorithms, either radix-2 ($5N \log_2 N$ flops), radix-4 ($4.25N \log_2 N$ flops) or radix-8 ($4.08N \log_2 N$ flops), but these algorithms are not a general solution. The 'split-radix' algorithm [21] has a lower operation count ($4N \log_2 N$ flops) than either radix-4 or radix-8 FFTs but does not have the addressing regularity. The 'split-radix' method would be the algorithm of choice for a software solution if a single radix algorithm were acceptable, as any extra-storage needs (to indicate which 'butterflies' – the well-known characteristic data exchange step of the recursive binary algorithm – are needed and when) are easily absorbed. It is applicable to the same problem sizes as standard radix-2 algorithms, but still for large images it limits choice (there would be no FFT between size 512 and 1024) unless a compromise on speed is made by the use of zero padding. Therefore, the decision was made to implement a mixed-radix FFT.

The mixed-radix algorithm is an extension of (5) to the a-dimensional case [22, 17]:

$$\begin{aligned} X_{b+1}(u, v_{a-b}, v) &= \omega_{P(b)}^{uv_{a-b}} \sum_{u_{a-b}=0}^{P_{a-b}-1} X_b(u, u_{a-b}, v) \omega_{P_{a-b}}^{u_{a-b}v_{a-b}}, \qquad (9) \\ u &= u_0 + u_1 P_0 + u_2 P_0 P_1 + \ldots + u_{a-b-1} P_0 \ldots P_{a-b-2}, \\ v &= v_{a-b+1} + v_{a-b+2} P_{a-b+1} + v_{a-b+3} P_{a-b+1} P_{a-b+2} + \ldots + v_{a-1} P_{a-b+1} \ldots P_{a-2}, \end{aligned}$$

with $P(b) = P_0 \ldots P_{a-b}$ and $X_1(\cdot) = x(\cdot)$. Notice that: the recurrence can be programmed as a loop; at each stage of the recurrence the dimensions of array X change with index b (implemented by suitable indexing into a fixed-size 1-D array); and the summation on the R.H.S. constitutes an inner loop that can be unrolled (by small-order transforms). The RC algorithm has the happy property that 1-D mixed-radix algorithms based on a selection of optimized small-order transform components can directly be plugged into each row and column computation directly. Pseudo-code for small-order transforms can be found in an appendix to the seminal work of [23] where the routines are proven to be optimal in the number-theoretic sense [24].

The mixed-radix algorithm results in computational savings over the single-radix case.

Table 1 is a simple illustration of the relative savings, accurately predicted by earlier authors [25], which accrue by effectively altering the base of the logarithm, b, in the expression for the time complexity of the 1-D FFT ($\gamma N \log_b N$), where γ is a scaling factor. The base of the logarithm, b, is also the radix of the transform, while the logarithmic factor itself gives the number of stages in a transform. There are less stages in a higher radix transform, and each of those stages can be made more efficient by judicious restructurings, e.g. radix-8 complexity can be rewritten as $12.25N \log_8 N$ flops. For a size 256 FFT in theory one expects a 33% saving in real multiplications and a 9% saving in additions, whereas Table 1 shows a 34% saving for four processors, which is of similar magnitude. The remaining decline in efficiency can be attributed to the overheads from using four processors. The size 512 FFT (forward transform using double-sized floating point (f.p.) representation) is also compared in Table 1 and the savings over a radix two algorithm are predicted to be 28% for multiplications and 9% for additions, which is of similar magnitude to the actual saving for four processors. In Table 2 a more complete picture (for a later version of the code) is shown. The incorporation of radix-6 transforms improves the relative timings for larger image sizes. However, even for the larger images the efficiency declines noticeably after 8 processors. The timings for one transputer were with a different compiler and therefore may exaggerate the speed-up (all transputers ran at 25 MHz with links operating at only 10 MHz). Despite the presence of an 'on-chip' floating point unit the timings appear to reflect the weak f.p. operation of the processor. Largest savings in time occurred either when it was possible to use radix-four exclusively, when the twiddle factors are ± 1, or from radix-six operation by means of the Prime Factor Algorithm (PFA) [26].

The PFA makes use of the identity $RF_PC = F_{P_0} \otimes F_{P_1}$, where F_P is the Fourier matrix (i.e. $\{e^{ink2\pi/P}\}$ of (1) in matrix form), $P = P_0P_1$, $\gcd(P_0, P_1) = 1$ and $\otimes$ represents the Kronecker product, thereby avoiding scaling by a twiddle factor vector. C and R are permutation matrices forming the Chinese Remainder Theorem mapping and the Ruritanian mapping [27]. When the size of the (small-order) FFT is known in advance it is no longer necessary to pre-calculate the index permutations needed for the PFA algorithm.

Since the memory requirement at the processors executing the FFT engines is storage for one row/column, in-place computation is not a necessity, thus no longer requiring digit-reversal routines which are a not inconsiderable time overhead (Section 4). (The returning rows *are* stored in an in-place fashion.) These requirements funnelled attention upon Stockham's self-sorting algorithm [28], which digit reverses (9) at each stage of the recurrence, for the FFTs of individual rows. If the C programming language is used, no transfer of data between the row-storage vectors need occur, as the routines can alternate between the two row-storage vectors by pointer swapping.

2.4 The Centralized Phase

After the image rows have been processed there is a centralized stage when preparation is made to distribute the columns. The chief disadvantage of the RC algorithm is this centralized phase. Without loss of generality, assume row-major addressing. The current message-passing primitives provided by communication harnesses require a start address and a length. For this reason, viewing the image array as a matrix, it may be necessary to transpose it. This was found to be done most efficiently on the transputer-based machine as soon as the data arrived back (as row distribution was not delayed). There are also two ways of swapping the data around the main diagonal of the matrix if a separate routine is used after the row distribution phase. For machines using a memory hierarchy, as do most general purpose machines, a block-by-block transpose is also available as a way of avoiding cache misses [29]. However, our practical tests with different transpose algorithms on SPARCs recorded no advantage from a block-by-block method [30]. PVM has a communication mode which allows direct transfer of the arriving data items, but in general Unix socket-based implementations of PVM communication through memory-to-memory copies and 'mailbox' message-buffering may obviate any gains.

A way to reduce the sequential bottleneck involved in transposition might be to take advantage of a recursive routine [31] originally developed for large arrays (or alternatively restricted memory machines) as the basis of a parallel decomposition. We found that the

need to redistribute blocks of data did not in practice make this approach an attractive prospect. This was the case when the blocks were distributed by way of a tree processor topology, retaining some work at its root.

2.5 Handling Real-valued Data

Though an FFT specifically for real data was not provided, in retrospect two sets of FFTs would have been a preferable design decision. In [32], when appropriate comparisons are made, it is shown that such FFTs are at least as good as (say) the Hartley Transform for real-valued data.

For real-valued data it is possible to pack two successive rows (at load time) into the real and imaginary parts of one complex vector and recover two complete transforms using Hermitian symmetry ($F(k) = F^*(-k)$, where F is the Fourier transform operator, $k = 1, 2 \ldots N$ and the suffix * as usual denotes conjugacy. The advantage of this arrangement for parallel implementations is a saving in communication cost, since two transforms are accomplished from sending out one complex row vector. Once the packed data vectors had been processed it was found to be quicker to unpack the data in parallel at return time, even though more data were transmitted when returning the rows to the data-farmer. A greater virtue (by reason of Amdahl's law) was to reduce thereby the amount of centralized processing. Returning rows may be in the wrong order. One option, if the data-farmer process can do so without holding up work-scheduling (as with transposition), is to put the rows in the correct positions as they arrive back. Otherwise, the rows can be sorted without any data movement by using a tagged array. Since the row identities form a uniquely-ordered set, a bin sort, which is of time complexity $O(N)$, is suitable. If a transpose is performed, it must now be done by means of tag indexing.

As the complete transformed image (for real-valued data) exhibits Hermitian symmetry, only one half of the columns need be sorted, transposed and sent out for the last phase of the algorithm. Special treatment is needed for the zeroth and $N/2^{\underline{th}}$ columns. Notice that, though the transformed image is in transposed form, it is unnecessary to transpose back provided

the frequency domain processing software is cognizant of this fact, as the succeeding reverse transform will also reverse the transpose. The reverse transform offers less opportunity for reducing the computation (and amount of messages passed in a parallel version) but, since the final target image may be real-valued, pairs of columns can be packed into one complex vector. The column occupying the imaginary part of the vector must be pre-multiplied by i and the final reconstruction (including normalization) proceeds in parallel making using of the linearity of the FFT.

2.6 Other Considerations

The reduction of operations in the FFT does not generally reduce quantization noise since the set of operations on a single point are reorganised but not changed.[1] One must avoid rounding errors from repeated calculations. For example, a recursive method of calculating the rotation or 'twiddle' multiplication factors which only makes partial reference to library mathematical functions carries this risk.

For VLSI applications, the advantage of an in-line method of trigonometric generation is that extra storage is not required. For general parallel applications, a look-up table (LUT) can be created at run time for later use by individual processes. The creation time of the LUT, replicated on each process, can be offset against the image loading time on a root process. On the transputer without hardware support for virtual memory, which is booted-up with the complete program code (or on other embedded systems booted from ROM) a large LUT as well as the existing run-time kernel and any dynamic memory creation can exhaust the memory. Indeed, we had such problems when providing the tangent LUT mentioned in Section 3, despite 4M bytes of memory (the central farmer was privileged with 8M bytes).

Several forms of twiddle factor generation (library routine, in-line generation or LUT possibly using an in-line generation method) are part of a flexible solution which ideally

[1] It is assumed that the user of the FFT can select between single and double length f.p. representation so as to take advantage of the tradeoff between speed and accuracy. The transputer for instance has a 20:7 speed ratio between double and single f.p. operation.

should be provided by a command line switch so that recompilation is not necessary. In [33], there is a binary recursive method that has $O(u \log_2 j)$ accuracy where u is the unit error that might arise from an individual f.p. operation (i.e., $(x \text{ op } y)(1+\epsilon), \epsilon < u$) and j counts the order of generation. Unlike recursive methods that use constant multiplicative factors, the binary recursive method employs interpolation from a small, previously-stored LUT. In general, the size of any table should certainly be reduced (twofold) by using conjugate symmetry and reduced possibly fourfold at a cost in addressing. There is also a hybrid method of twiddle factor generation given in [34], which dynamically forms blocks of the twiddle table (when viewed as a 2-D array) from a basic block and a number of fixed elements. This method reduces storage to $8\sqrt{N}$ but involves four complex multiplications for each twiddle factor generated, which implies rounding errors.

An added flexibility in all complex-field algorithms is to provide alternative methods of complex multiplication as, by means of Golub's algorithm [25], the ratio of four to two for multiplication to add can be switched to three to five. Examination of the instruction cycle times for the transputer shows that as multiplication and addition times are similar this alternative scheme would not lead to a timing improvement for us.

A problem with routines adapted from FORTRAN [35] is that complex vectors are stored as alternate real and imaginary parts rather than separate vectors. On machines with interleaved memory (some multiprocessors) this is likely to cause difficulties because the stride is no longer one. In writing code, it provides practical problems in keeping track (and updating) two sets of indices. The disadvantages of FORTRAN vectors make it preferable to send the real and imaginary vectors separately despite the extra set-up time needed. Finally, 'Iliffe' vectors whereby data access is from an array of row-vector pointers avoid data movements when performing the unscrambling steps needed for the implementations of Section 4.3.

3 Extending to a DCT

The programmer will want to economize on the effort in developing software by reusing sound code. In this section, we reiterate how this can be done in the case of the DCT and indicate the main computational problem with such an approach. The 2-D DCT uses an adapted FFT, as first described in [36]. This DCT does not rely on zero padding a double length FFT transform, but keeps the length the same by judicious use of various symmetries. Define the 2-D FFT as follows:

$$V(k_1,k_2)=\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}v(n_1,n_2)W_{N_1}^{n_1k_1}W_{N_2}^{n_2k_2}, \tag{10}$$

with $0\le k_1\le N_1-1,\ 0\le k_2\le N_2-1$, $W_N=e^{\frac{-2\pi i}{N}}$, $v(n_1,n_2)\in R$, where R is the set of reals. It should be noted that $W_N^n=W_{kN}^{kn}$ for $k\in Z$. Consider

$$C(k_1,k_2)=2\Re\left(W_{4N_1}^{k_1}\left(W_{4n_2}^{k_2}V(k_1,k_2)+W_{4n_2}^{-k_2}V(k_1,N_2-k_2)\right)\right). \tag{11}$$

Use (10) to expand (11):

$$\begin{aligned}
C &= 2\Re\left(W_{4N_1}^{k_1}\left(\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}v(n_2,n_2)W_{N_1}^{n_1k_1}W_{4N_2}^{(4n_2+1)k_2}+\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}v(n_1,n_2)W_{N_1}^{n_1k_1}W_{4N}^{-(4n_2+1)k_2}\right)\right)\\
&= 4\Re\left(\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}v(n_1,n_2)\left(\cos\left(\frac{\pi(4n_2+1)k_2}{2N_2}\right)\cos\left(\frac{\pi(4n_1+1)k_1}{2N_1}\right)+i(\cdot)\right)\right) \qquad (12)\\
&= 4\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}v(n_1,n_2)\cos\left(\frac{\pi(4n_2+1)k_2}{2N_2}\right)\cos\left(\frac{\pi(4n_1+1)k_1}{2N_1}\right), \qquad (13)
\end{aligned}$$

where C represents the particular form of DCT used here and where we use $\mathcal{R}$ and $\mathcal{I}$ respectively to select the real and imaginary parts of a complex number.

A difficulty with this scheme is that the post-multiplications after the FFT can swallow up the advantage of using an FFT approach, especially for small data length transforms. The multiplications can be schematized as two rotations and an addition of terms. This produces the results for two columns at once from one column using conjugate symmetry. The rotation matrix used is:

$$\mathbf{R}(\mathbf{k_1},\mathbf{k_2})=\begin{bmatrix}\cos\left(\frac{2\pi(k_1+k_2)}{4N}\right) & -\sin\left(\frac{2\pi(k_1+k_2)}{4N}\right)\\ \sin\left(\frac{2\pi(k_1+k_2)}{4N}\right) & \cos\left(\frac{2\pi(k_1+k_2)}{4N}\right)\end{bmatrix}, \tag{14}$$

with $N = N_1 = N_2$. Then

$$\begin{bmatrix} \mathrm{DCT}(k_1,k_2) \\ \mathrm{DCT}(N-k_1,k_2) \\ \mathrm{DCT}(k_1,N-k_2) \\ \mathrm{DCT}(N-k_1,N-k_2) \end{bmatrix} = \mathbf{P}.\begin{bmatrix} \mathbf{R}(\mathbf{k_1},\mathbf{k_2}) & \mathbf{0} \\ \mathbf{0} & \mathbf{R}(\mathbf{k_1},-\mathbf{k_2}) \end{bmatrix} \begin{bmatrix} \Re(\mathrm{DFT}(k_1,k_2)) \\ \Im(\mathrm{DFT}(k_1,k_2)) \\ \Re(\mathrm{DFT}(k_1,N-k_2)) \\ \Im(\mathrm{DFT}(k_1,N-k_2)) \end{bmatrix}, \quad (15)$$

where $\mathbf{0}$ is a 2×2 zero-valued matrix and the permutation matrix is

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix}. \quad (16)$$

The multiplication matrix can be decomposed further in the manner of the thrice-skew rotation algorithm, resulting in only 3 multiplications and 3 additions. Put $\theta = \frac{2\pi(k_1+k_2)}{4N}$. Then

$$\mathbf{R}(\mathbf{k_1},\mathbf{k_2}) = \begin{bmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin(\theta) & 1 \end{bmatrix} \begin{bmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{bmatrix}. \quad (17)$$

The advantage of using this method has to be balanced against the need to store and access the tangent LUT, since it cannot apparently easily be constructed from the existing sine and cosine tables necessary for the FFT. A further difficulty may arise from additional rounding errors.

On two trees of three transputers, a 256×256 pixel image took 2.26s and 4.12s respectively for forward and reverse double precision f.p. DCT, making use of data packing and Hermitian symmetry.

4 Convolution for Large Images

In this section, we examine a block-by-block method of parallel convolution, adapted from restricted memory serial implementations. The method can still use data-farming, as for the

FFT and DCT previously discussed, but the farmed items are now blocks or tiles not rows of the image. The problem with this approach, which is seemingly unavoidable, is forming the blocks before transmission because of non-consecutive memory addressing, which will perturb the memory hierarchy.

4.1 Parallel Overlap-Save Convolution

In some circumstances it is preferable to take advantage of frequency-domain convolution, even though this can involve both a forward and reverse transform. The computational complexity of spatial-domain convolution is $O(N^2M^2)$ multiplications, assuming square image of size N and square window of size M, though this can be ameliorated by grouping common calculations, by arranging for mask coefficients to be zero or unity or by using shift operators. To find the computational advantage of using a spatial method rather than a frequency-domain method of convolution, the following inequality must be approximately followed:

$$M^2N^2 < \gamma 2(N+M-1)^2 \log_b(N+M-1) + (N+M-1)^2 \tag{18}$$

from which a rule of thumb is:

$$M^2 < 2\log_2 N. \tag{19}$$

It is usual to pre-compute the mask transform and store it as an LUT.

A problem with using a frequency-domain method is that the memory usage is large. In a serial implementation, both the augmented image and mask must be stored. For a parallel implementation, one might consider storing the complete transformed mask on each worker in pre-calculated form. If instead the transformed mask is split between processors, one must arrange for the matching part of the transformed image to arrive at the appropriate processor. Compared to a spatial method, when the same mask is stored on each processor, the data loading time is a substantial disadvantage. To avoid the problem of too large borders as well as the data distribution problem, an alternative is to use a block method: as the image is split into blocks, the transformed mask is only the size of a block. The advantage for a parallel

method is that only a small mask need be stored or transmitted to the worker processor. The block method is computationally less efficient since the size of the transform has been reduced (and the computational complexity of the FFT reduces logarithmically) but for very large images it might not be possible to load the whole of the image into main memory.

As linear convolution is usually required whilst the FFT gives circular or wrap-around convolution it is necessary to enlarge the border region of each block so that its size (assuming square blocks and square masks) becomes $B+M-1$, where B is the original size of the block and M the original size of the mask. One can either augment each block by a zeroized strip or re-use block border data for the strip. In the former method (overlap and add), the processed block is added to the existing data. For a serial implementation, this means source and target images are required. For a parallel implementation, one must avoid returning data overwriting data yet to be sent. If the existing border of each block is used (overlap and save), the surround is discarded by the worker process before returning it. A bonus is that the size of data returning is reduced. Additionally, the need to take time in adjoining any zeroized block borders is avoided, as of course are the additions of the returning block border edges. As returning blocks are offset from the original blocks (Figure 2) they were pushed up and back to avoid overwriting blocks yet to depart, a zeroized image border providing a convenient cushion to allow this (Figure 3).

4.2 Organization of the Algorithm

Two varieties of convolution program are possible, with and without bit-reversal.

Assuming a radix-2 (or radix-4) implementation, if one starts with a forward decimation in time (DIT) transform and (after the element-wise multiplication) use a reverse decimation in frequency algorithm (DIF) bit-reversal is not needed. Notice that the PT algorithm does not allow this possibility unless one uses bit-reversed tagging of data.

The bit-reversal algorithm will usually scan the data index range establishing the bit-reversed version by use of Horner's algorithm. Optimized programs on various DSPs took between 3 and 15% of the time in [37], which is still not inconsiderable. For software-based

implementations one of the fast bit-reversal algorithms [38], which use a recursive method to avoid unnecessary scanning of the data index range, will reduce the index permutation calculation but data swapping remains as a fixed overhead.

To cope with (say) a radix-two algorithm the block size and/or image size can be increased. If the block size (before augmentation) does not fit the image exactly then the image is also incremented on its far side. There are two situations that can arise. The remainder portion might not be enough to augment the block, in which case an extra strip is added. Alternatively, the remainder may be too large to augment the block, in which case an extra block must be made up with dummy values. This is shown in Figure 4. Using zeroes as the dummy values on the borders results in the average intensity being reduced for those pixels adjacent to the zeroized region and some other choice of border grey-level might be considered. For example, if a mask size of 10 were to be used with an image block size of 54 were employed, then the required total size of 63 is made up to 64 to enable a radix-2 transform to be used. The image size is augmented from 512 to 522 to give a zeroized border. When the augmented image size is sampled in unaugmented blocks of size 54 then 9 blocks can be fitted in with a remainder of 36. But an edge of only 10 is required. Thus, an extra block is required for the last 16 pixels. The previous 10 pixels are re-used and the 16 pixels must be made up to 54 by zeroizing a final strip. The image must be increased in a vertical as well as horizontal fashion to account for fractional block sizes. A detailed consideration of optimal block size is given in [39]. Choice of image size is also relevant for larger images if there is a fractional extent [40].

4.3 Choice of Block FFT

Despite the advantage of the PT algorithm in reducing multiplication operations and thereby increasing accuracy and also the theoretical 25% computational advantage of the VR algorithm over the radix-2 RC algorithm, timings revealed that the RC algorithm was faster in typical circumstances. Tables 3 & 4 contains sample timings. In Table 3, the image/block sizes are selected merely for a comparative timing test. In Table 4, the timings were made on a 50

MHz SPARCstation 5.

4.4 Implementation with the RC Algorithm

The RC algorithm on a per-block basis can use data packing (Section 2.5) in combination with bit-reversal avoidance but then it is necessary to perform any operations on the normal order of the data, not on their bit-reversed versions. A way around this is to use tags which have been pre-sorted into bit-reversed order. Then, by indexing through the tag, the normal order can be accessed. Clearly for small vectors this will obviate the advantage of packing in the first place. Similar savings by means of another form of data packing (Section 2.5) can be taken advantage of in the inverse transform for each block. The resulting sequence of operations is shown in Figure 5, which summarizes the parallel implementation.

5 Conclusion

This paper has given an overview of the implementation of frequency transforms for a parallel software library suitable as an accelerator to dsp software. Though there are a large number of schemes proposed for FFTs and associated routines, for a portable and parallel implementation, the Row-Column (RC) 2-D algorithm forms the most appropriate basis. Similarly, a mixed-radix autosort routine for the individual FFTs is flexible and in practice is competitive in computational terms. The RC algorithm can be adapted for a 1-D transform and for a DCT. The efficiency of the algorithm is also as much a function of auxiliary routines, such as bit-reversal or matrix transpose, as it is of the organization of the 'butterflies'. The speed of the matrix transpose may be affected by the architecture of the machine on which it is performed. As this forms the bulk of the centralized processing in a parallel RC implementation, it may well call for user-selectable methods when supplied as part of a software library. Selection of the mode of generation of trigonometric values is also advisable. For real-valued data there are also a number of simple ways that data communication and processing can be reduced. To implement these methods, which were adopted in all the transforms described

here, requires careful 'bookkeeping'. Similar logistics are required when performing parallel convolution by a block method when there may also be the problem of augmented or fractional blocks.

Acknowledgement

We thank the anonymous referees for their stimulating comments. This work was carried out as part of project IED3/1/2171 ("Parallel Reconfigurable Image-Processing Systems") and part of EPSRC research contract GR/K40277 'Parallel software tools for embedded signal-processing applications' as part of the EPSRC Portable Software Tools for Parallel Architectures directed programme.

References

[1] A. F. Clark. PRIAM: A demonstrator for PRIPS, design issues. Technical report, University of Essex, 1994.

[2] G. E. Rivard. Direct Fast Fourier Transform of bivariate functions. *IEEE Transactions On Acoustics,Speech, and Signal Processing*, 25(3):250–252, 1977.

[3] H. J. Nussbaumer. Two-dimensional convolution and DFT computation. In T. S. Huang, editor, *Two Dimensional Digital Signal Processing II Transforms and Median Filters*, pages 37–89. Springer, Berlin, 1981.

[4] D. E. Dudgeon and R. M. Mersereau. *Multidimensional Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, 1984.

[5] G. L. Anderson. A stepwise approach to computing the multidimensional Fast Fourier Transform of large arrays. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(3):280–284, June 1980.

[6] H. Kunieda and K. Itoh. Parallel 2D-FFT algorithm on a practical multiprocessor system. In D. May and K.L. Kunii, editors, *Proceedings of the* $3\underline{^{rd}}$ *Transputer-Occam International Conference*, pages 77–89. IOS, Amsterdam, 1990.

[7] Y. Huang and Y. Paker. A parallel FFT algorithm for transputer networks. *Parallel Computing*, 17:1–12, 1991.

[8] J. J. Dongarra, G. A. Geist, R. Manchel, and V. S. Sunderam. Integrated PVM framework supports heterogeneous computing. In A. Y. Zomaya, editor, *Parallel Computing: Paradigms and Applications*, pages 435–454. Thomson, London, 1996.

[9] M. Fleury, L. Hayat, and A. F. Clark. Parallel reconfiguration in an image-processing context. *Concurrency: Practice and Experience*, 1997. Accepted for publication.

[10] M. Fleury, L. Hayat, and A. F. Clark. Parallelizing grey-scale coordinate transforms. *IEE Proceedings Part I (Vision, Image and Signal Processing)*, 142(4):207–212, August 1995.

[11] U. Rüde. Iterative algorithms in high performance architectures. In *Euro-Par'97*, pages 57–71. Springer, Berlin, 1997. Lecture Notes in Computer Science Vol. 1300.

[12] P. Ling. Circumventing the cycle. *New Electronics*, pages 30–31, 1997.

[13] 3L Ltd., 86/92 Causewayside, Edinburgh EH9 1PY, Scotland. *Parallel C User Guide*, 1991.

[14] H. Sava, M. Fleury, A. C. Downton, and A. F. Clark. Parallel pipeline implementation of wavelet transforms. *IEE Proceedings Part I (Vision, Image, and Signal Processing)*, 1997. In press.

[15] D. M. Harvey, S. P. Kshirsagar, C. A. Hobson, D. A. Hartley, and J. D. Moorehead. Digital signal-processing systems architectures for image processing. In $5\underline{^{th}}$ *International Conference on Image Processing and its Applications*, pages 460–464, 1995. IEE Conf. Publ. No. 410.

[16] M. C. Pease. An adaptation of the fast Fourier transform for parallel processing. *Journal of ACM*, 15:252–264, 1968.

[17] P. N. Swarztrauber. Multiprocessor FFTs. *Parallel Computing*, 5:197–210, 1987.

[18] A. Averbuch, E. Gabber, B. Gordissky, and Y. Medan. A parallel FFT on an MIMD machine. *Parallel Computing*, 15:61–74, 1990.

[19] A. S. Wagner, H. V. Sreekantaswamy, and S. T. Chanson. Performance models for the processor farm paradigm. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):475–489, May 1997.

[20] M. Fleury, A. C. Downton, and A. F. Clark. Modelling pipelines for embedded parallel processor system design. 1997. Submitted to Electronic Letters.

[21] P. Duhamel. Implementation of "Split-Radix" FFT algorithms for complex, real, and real-symmetric data. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(2):285–295, April 1986.

[22] C. Temperton. Self-sorting mixed-radix Fast Fourier Transforms. *Journal of Computational Physics*, 15:1–23, 1983.

[23] S. Winograd. On computing the Discrete Fourier Transform. *Mathematics of Computation*, 32(141):175–199, 1978.

[24] S. Zohar. Winograd's discrete Fourier transform algorithm. In T. S. Huang, editor, *Two Dimensional Digital Signal Processing II Transforms and Median Filters*, pages 89–160. Springer, Berlin, 1981.

[25] R. C. Singleton. An algorithm for computing the mixed radix fast Fourier transform. *IEEE Transactions on Audio and Electroacoustics*, 17(2):93–103, 1969.

[26] C. S. Burrus and P. W. Eschenbacher. An in-place, in-order prime factor FFT algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(4):806–817, 1981.

[27] I. J. Good. The interaction algorithm and practical Fourier analysis. *Journal of the Royal Statistical Society Series B*, 20:361–372, 1958.

[28] M. L Uhrich. Fast Fourier Transforms without sorting. *IEEE Transactions on Audio and Electroacoustics*, pages 170–172, 1969.

[29] C. van Loan. *Computational Frameworks for the Fast Fourier Transform*. SIAM, Philadelphia, 1992.

[30] M. Fleury, H. Sava, A. C. Downton, and A. F. Clark. A real-time parallel image-processing model. In $6^{\underline{th}}$ *International Conference on Image Processing and its Applications*, volume 1, pages 174–178, 1997. IEE Conference Publication No. 443.

[31] J. O. Eklundh. A fast computer method for matrix transpose. *IEEE Transactions on Computers*, pages 801–803, 1972.

[32] H. K. Sorensen, D. L. Jones, M. T. Heidman, and C. S. Burrus. Real-valued Fast Fourier Transform algorithms. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(6):849–863, 1987.

[33] O. Buneman. Stable on-line creation of sines and cosines of successive angles. *Proceedings of IEEE*, pages 1434–1435, 1987.

[34] D. H. Bailey. FFTs in external or hierarchical memory. *Journal of Supercomputing*, 4:23–35, 1990.

[35] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. C.U.P., Cambridge, UK, 1988.

[36] J. Makhoul. A Fast Cosine Transform in One and Two Dimensions. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(1):27–34, 1980.

[37] R. Meyer and K. Schwarz. FFT implementation on DSP-chips — theory and practice. In *Proceeding of the International Conference on ASSP*, volume 3, pages 1503–1506, 1990.

[38] J. Jeong and W. J. Williams. A fast recursive bit-reversal algorithm. In *Proceeding of the International Conference on ASSP*, volume 3, pages 1511–1514, 1990.

[39] B. R. Hunt. Minimising the computational time for using the technique of sectioning for digital filtering of pictures. *IEEE Transactions on Computers*, 21:1219–1222, 1972.

[40] R. E. Twogood, M. P. Ekstrom, and S. K. Mitra. Optimal sectioning procedure for the implementation of 2-dimensional digital filters. *IEEE Transactions on Circuits and Systems*, 25:260–268, 1978.

List of Figures

List of Tables

No. of active processors	Size	Binary (2)	Mixed (2,4)	Saving %
4	256	4.36	2.89	34
4	512	20.20	13.43	33
8	256	3.21	2.23	31
8	512	14.46	9.96	31
10	256	2.32	1.95	16
10	512	9.96	7.65	23
12	256	2.26	1.98	12
12	512	8.92	7.53	16

Table 1: Recorded Savings (s) between Mixed-Radix and Binary FFT using T805 Transputers

Image size	Factorization	No. of Processors					
		1	4	6	8	10	12
128	4,4,4,2	4.18	0.63	0.56	0.53	0.54	0.56
144	6,6,4	5.01	0.87	0.68	0.67	0.66	0.69
216	6,6,6	11.54	2.01	1.49	1.40	1.38	1.40
256	4,4,4,4	18.02	2.89	2.23	1.99	1.95	1.98
288	6,6,4,2	25.20	4.18	2.97	2.55	2.50	2.47
384	6,4,4,4	43.40	7.30	5.16	4.41	4.29	4.23
432	6,6,6,2	60.40	9.74	6.84	6.72	5.43	5.23
512	4,4,4,4,2	88.36	13.43	12.02	8.30	7.65	7.53
576	6,6,4,4	105.33	16.18	12.02	10.05	9.59	9.13

Table 2: Execution Timings (s) for a Mixed-Radix Real-Valued 2D-FFT using T805 Transputers

Image-Block Size	Filter-Block Size	Time (s)	Data Type	Algorithm
32	32	25.65	Real	R/C
32	32	34.96	Complex	R/C
32	32	44.45	Real	Poly
32	32	46.20	Complex	Poly
54	10	11.33	Real	R/C
54	10	15.83	Complex	R/C
54	10	18.43	Real	Poly
54	10	19.39	Complex	Poly

Table 3: Comparison of Algorithms for Convolution using a size 512 image with 9 Transputers

Image size	Algorithm	Real-valued Data Time (s)	Real-valued Data Difference %	Complex-valued Data Time (s)	Complex-valued Data Difference %
32	RC	0.019	24	0.023	8
32	VR	0.025		0.025	
64	RC	0.079	37	0.094	23
64	VR	0.126		0.123	
128	RC	0.408	35	0.499	14
128	VR	0.624		0.581	
256	RC	1.715	41	3.086	
256	VR	2.888		2.856	7
512	RC	8.101	37	9.896	40
512	VR	12.909		13.855	

Table 4: Comparison of Algorithms for 2-D FFT (Serial Version on SPARCstation 5)

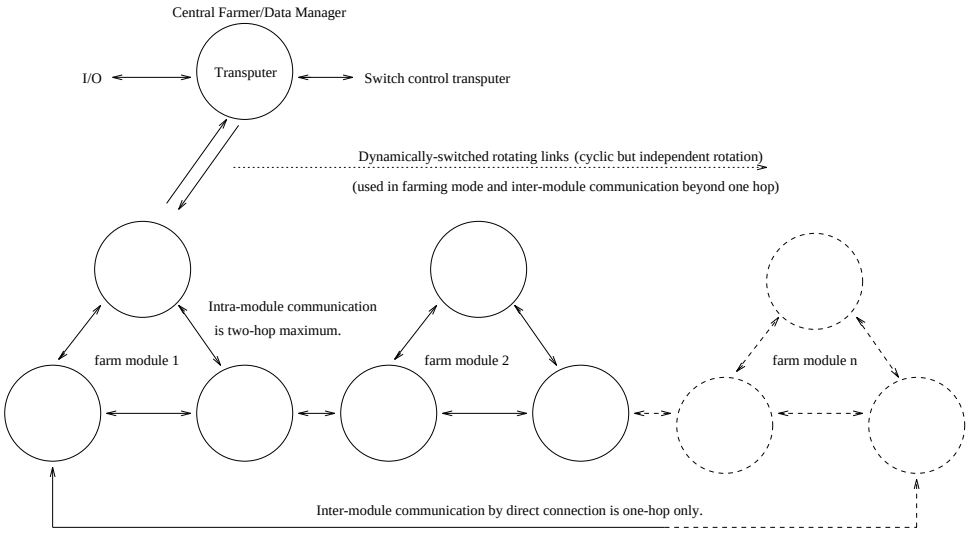

Figure 1: Generic Hardware Structure Implemented on the Parsys Supernode

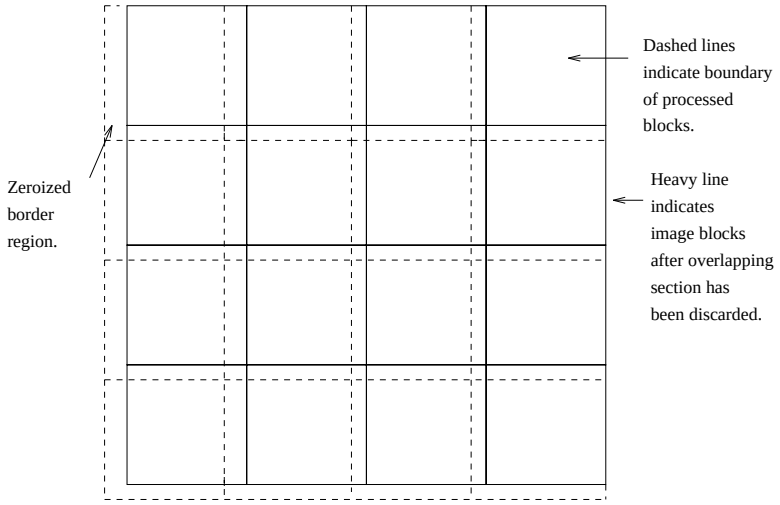

Figure 2: Block Allocation for the Overlap-Save Method

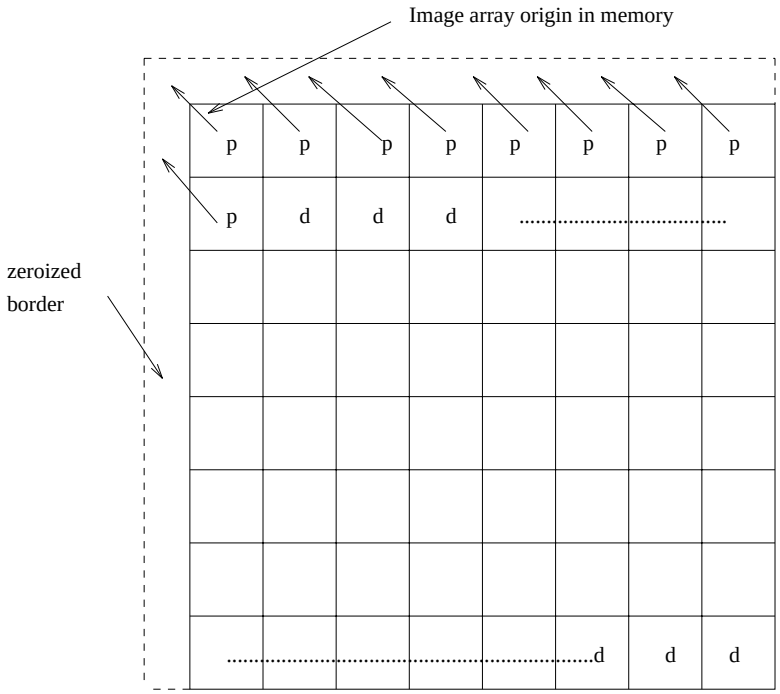

Figure 3: Saving Returning Blocks without Overwrite

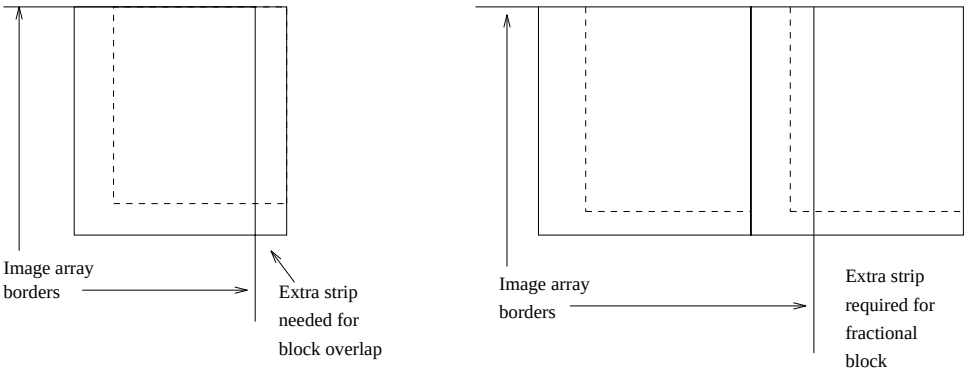

Figure 4: Coping with Fractional Blocks

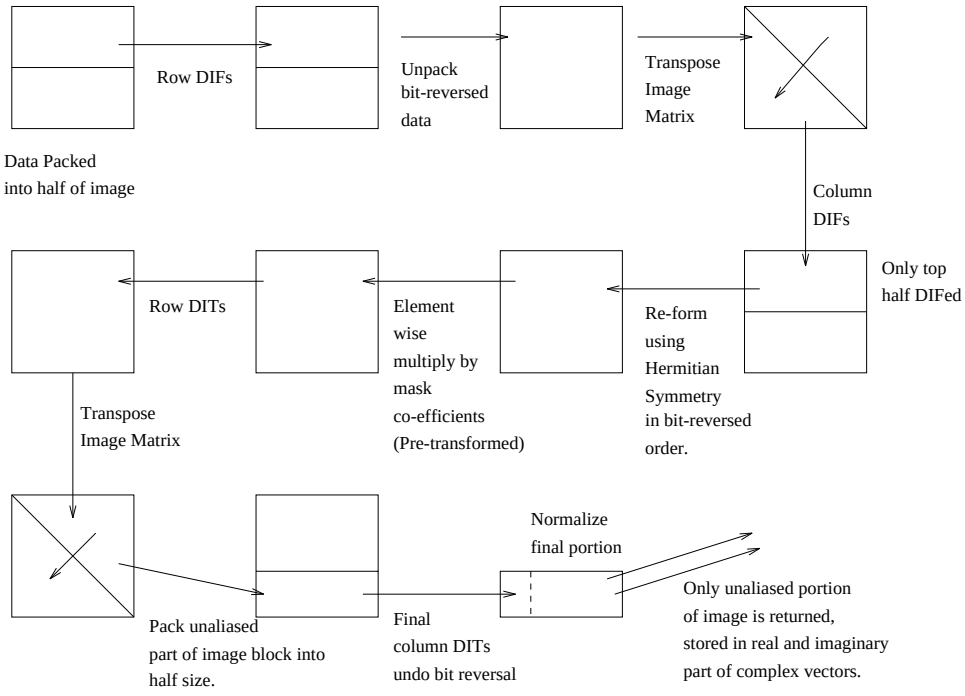

Figure 5: Processing Real-Valued Data