October 1976

VLA COMPUTER MEMORANDUM NO. 132

An Algorithm for Mapping Selected Regions of the Field of View

by Jerry Hudson

It is often desirable to make a map of only a limited region within the field of view (defined, say, by the principal beam of the antennas) of a radio interferometer. Similarly, it might be desired to examine the profile of a single spectral line feature when doing time-domain spectral analysis, when other features were within the band-pass of the recording instrument. Ignoring the possible presence of outside features while mapping a limited region with the Fast Fourier Transform (FFT) is to invite aliasing; direct evaluation of the equation

$$f(x_k, y_l) = \sum_{1=m}^{M} 2 \operatorname{Re} V_m \exp -2\pi j(u_m x_k + v_m y_l)$$

where the V's are calibrated measurements of the fringe visibility, u, v the baseline projections, and the set {k,l} of points on the x, y plane are assumed to form a square grid, K,L in size, requires some K.L.M iterations of an elementary step involving 6 multiplications and 3 additions. Even in microprocessors where the elementary steps could be kept down to, say, 1 usec, the size K.L of the region is severely limited by processing speed. (For 21,000 input data and a time limit of 10 minutes, the map size is limited to 169 X 169 points. In ordinary processors, where 60 usec is more typical, the size is cut to 22 X 22 points!)

An alternative to both of the above techniques is obviously in order. To this end, a modification of the FFT is suggested here, which should provide more reasonable computational speeds while avoiding aliasing. It would be required that the region of interest be a power of 2 in size, as with the conventional FFT.

We then proceed as follows:

1) First, the V(u,v) data must be sorted on, say, the u coordinate. Barry Clark has suggested that, for the VLA, the data from 351 baselines in a 10-second sample be initially sorted; the sorted sequences can then either be merged on demand, or automatically merged at fixed time intervals. Calibration would depend upon past history.

2) Next, the origin of the map is shifted by multiplying the visibilities V by a phase factor:

$$V' = V.\exp -2\pi j(u.x0 + v.y0)$$

where (x0, y0) is centered on the region of interest. This step can proceed concurrently with the following one.

3) As the V data are read from the sorted file, rows of width du in the (u, v) plane are constructed and transformed with a modification of the FFT, in the following manner. Suppose we have made du of such a size that 1/du is somewhat greater than the field of view (area of sensitivity on the sky) of the instrument. Suppose furthermore that we treat the (u, v) plane as consisting of N X N elements of size du X du, where $N = 2^n = Q.L$, L being the size of our region of interest, and Q of course also a power of 2, $Q = 2^q$, say. In the modified FFT, n passes through the data are needed, but many nodes in the "butterfly" diagram can be skipped, as seen in the examples in the figures. In fact, the rule for skipping nodes at pass i, i=n-q+1,n-q+2,...,n, is DO 2^{n-i}, SKIP $Q - 2^{n-i}$. We note that the outputs in locations 0, Q, 2Q, ... are in the bit-reversed sequence for the (n-q)-bit indices 0, 1, ..., L-1, and so bit-reversing takes place much as usual, except that the inputs are retrieved from every Qth cell in the row. Only the L outputs are written out on secondary storage, but it is necessary to write out all N rows, as these are transformed.

We note that the computation time required for this modified FFT goes approximately as $(1 + \log_2 L).N$, or roughly Q times the computation time for a row of length L.

(A word of explanation for the figures. At each node, the lines converging from the preceding pass on the left are understood to cause the inputs on the left to be added (complex addition). A dashed line indicates negation before adding. The symbol w is the nth root of unity; where w to some power appears, it multiplies the input.)

4) It should be noted that some effort is saved if instead of writing out N rows of length L complex numbers, we instead write N/2 + 1 rows corresponding to indices 0, 1, ..., N/2 in the u coordinate. We can thus take advantage of the Hermitian nature of the data by not bothering with the redundant half-plane. (I thank Larry D'Addario for suggesting a scheme for handling the half-plane without excessive data shuffling. The scheme now in use in our DEC-10 programs involves a modification of Larry's suggestion, which works as follows: The complex plane is taken to hold N/2+1 rows of length N complex numbers. After the row transforms, the half-length columns are modified according to the scheme:

$$C_k = 2^{1/2} [c_k + c^*_{n/2-k} + jc_k w^{-k} + jc^*_{n/2-k} w^{n/2-k}]$$

Following the convention, as we do, that real and imaginary components of a complex number are stored in successive storage locations, we have the convenient result that, after the FFT, the column of length N/2 contains the N real values for that row of the output map, stored in successive memory locations,)

5) Transposition of the array can be carried out by an algorithm which consists of transposing sub-matrices of size Nc X Nc, where $L = A.Nc^b$, A, b integers, where the rows of the matrices are fetched at intervals of Nc^p, $p = 0, 1, \ldots, b$, with an exception at the end (or better yet at the beginning) to handle the matrices of size A X A. The transposition thus requires $\mathrm{CEIL}(\log_{Nc}(L))$ passes for each L X L section of the L X N array. The algorithm resembles that of Eklundh (1972), although inspired by that of Knuth (1973).

6) We now do the column transform on what are now "rows" which we piece together from Q different places on the file (or Q different files). Again, we use the modified FFT in order to skip processing nodes which do not affect the desired L points in each row. Writing out the L points after bit-reversing, we are finished.

Barry Clark has suggested an alternative to the above algorithm, applicable if fast memory capable of holding $L^2(q+1)$ complex numbers is available.
Output on mass storage intermediate between sorted visibility data and the final map output is avoided. Sorting of the data is such as to place rows 0, 2.L, 4.L, 8.L, ..., , 1, 2.L+1, 4.L+1, ... in that sequence. We work from right to left on the butterfly diagram, calling upon a recursive procedure which, at pass i (i=0, 1, 2, ..., n) attempts to merge pairs of inputs from the left, according to the lines on the diagram. Here, inputs are entire rows, of length L, which have presumably been Fourier transformed in one direction. If the inputs are available, the procedure combines them, keeping one and discarding the other (provided i>n-q). If the inputs are not available, the routine calls itself recursively, putting i←i-1. If the routine reaches level n-q without obtaining inputs, it proceeds to read from the input file the rows 2.L+j, 4.L+j, ..., j being the row number belonging to the node sought. The rows are Fourier transformed in

the row sense, and then combined through n-q passes, as indicated on the butterfly diagram. In the worst case, one may have (q-1).L rows awaiting partners plus 2.L rows which have just been read in, for a total of (q+1).L rows.

For example, in Fig. 2, we would start at node 0, pass 3. Failing to find inputs 0 and 1 at pass 2, the procedure recurses, seeking an input at node 0, level 2 (call it input(0,2), say). Of course, the procedure immediately recurses again, since inputs (0,1) and (0,2) are unavailable. Now we reach level 1=n-q. The procedure therefore reads rows 0 and 4, Fourier transforms them, combines them according to the first pass indicated on the diagram, and then returns. Back at level 2, the procedure finds input (0,2) has been satisfied, but (0,2) is missing. Back to 1, this time seeking row 2. Two more inputs, rows 2 and 6, are obtained, transformed, and combined. Back at level 2, the procedure is at last satisfied, whence it now combines all pairs of level 1 inputs to make a level 2 node, discarding the lower of the two. (Thus we make (0,2) out of (0,1) and (0,2), discarding (0,2); (4,2) out of (4,1) and (6,1), discarding (6,1).) The procedure returns. Up at level 3 we discover input (2,1) is missing, whence another excursion which will not terminate until rows 1 and 5, then 3 and 7, are read.

Of course, recursion is not absolutely necessary in the implementation of this algorithm, but it aids the explanation.

I would like to thank Barry Clark for helpful discussion and criticism, and for encouraging me to find a way to reduce the computation of the FFT's.

References

Eklundh, J.O. (1972) A Fast Computer Method for Matrix Transposing, _I.E.E.E. Trans. on Computers_, _C-21_, 801--803.

Knuth, D.E. (1973) _The Art of Computer Programming_, Vol. 3, Addison-Wesley, Reading, Mass.

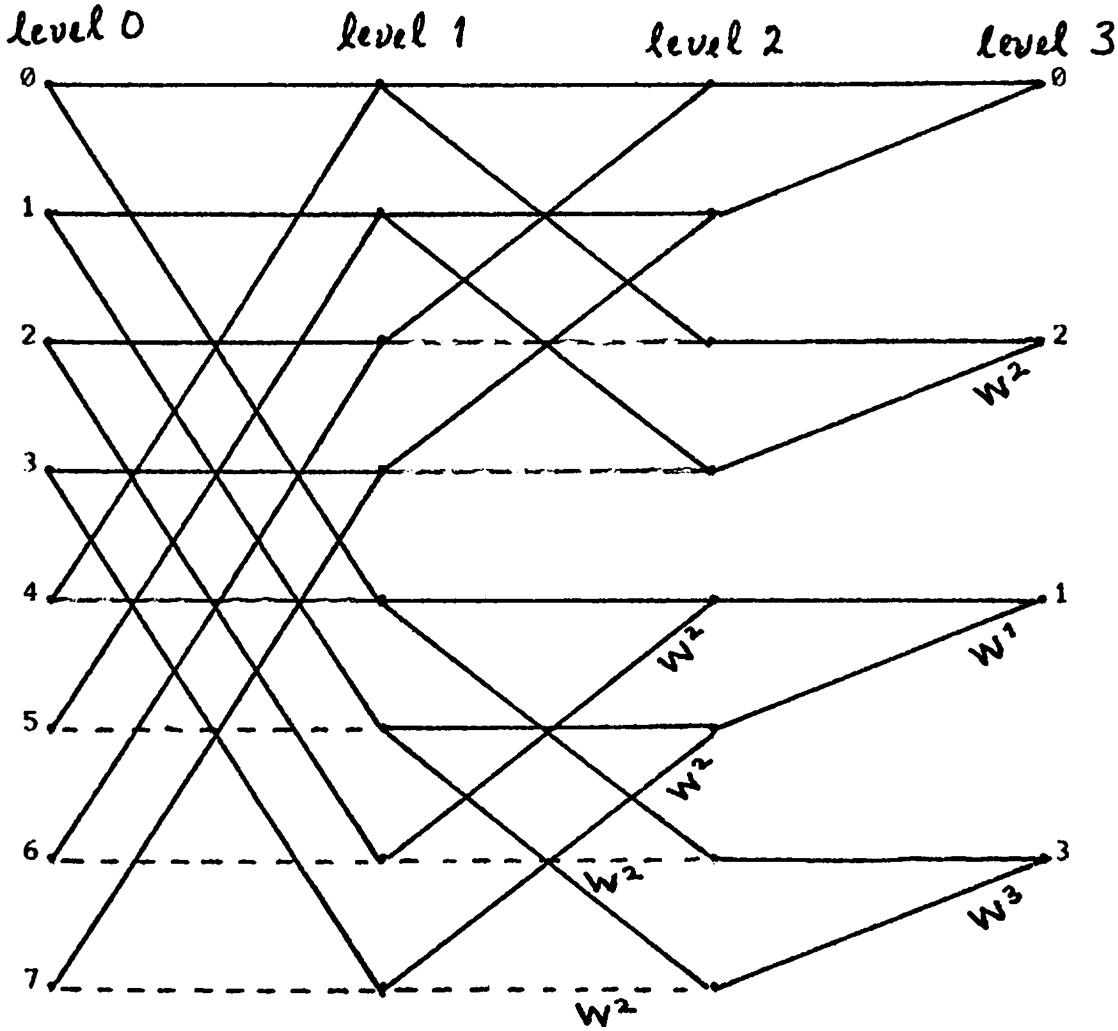

Fig. 1. Butterfly diagram for Q=2 (2:1 reduction)

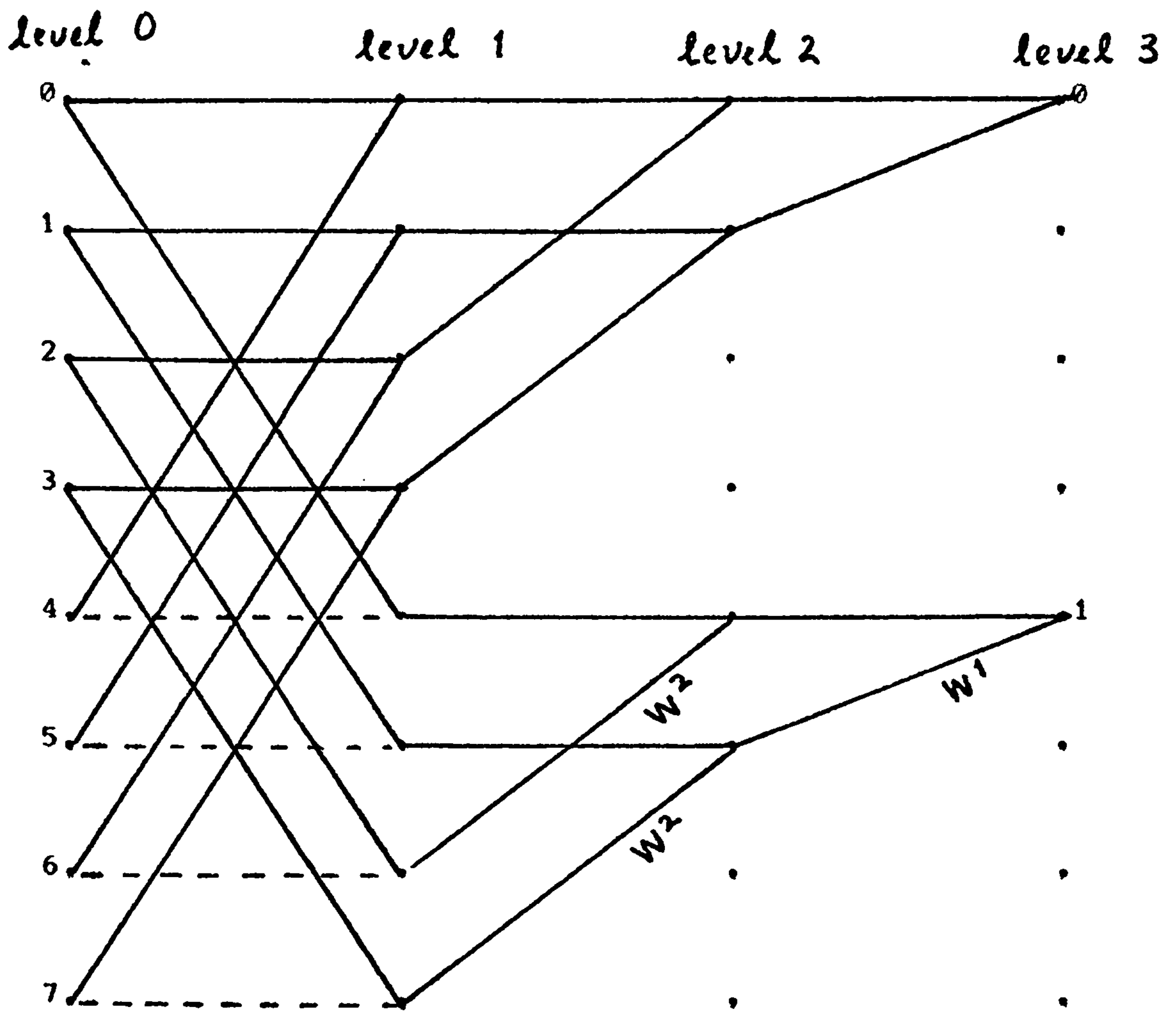

Fig. 2. Butterfly diagram for Q=4 (4:1 reduction)