James W. Cooley

Rectangular Transforms for Digital Convolution on the Research Signal Processor

The Rectangular Transform (RT) method for computing convolutions belongs to a family of Reduced Computational Complexity (RCC) algorithms. Convolution calculations by the RT method were programmed for the Research Signal Processor (RSP) and run on the RSP simulator, giving tabulations of numbers of RSP machine cycles. One of the original objectives was to see how well the original RSP architecture was suited to the RCC algorithms and to be able to make suggestions for possible changes. The results are also intended to demonstrate the efficiency of the RT convolution algorithms on a microprocessor with a limited instruction set and to show how to construct efficient RT programs for digital convolution. All results are given for the original RSP, as it was before the modification which resulted in the Real-time Signal Processor described in another paper in this issue [1].

Introduction and background

Considerable theoretical advances have been made in Reduced Computational Complexity (RCC) algorithms for some basic calculations such as the computation of multilinear forms, convolutions, and the Discrete Fourier Transform (DFT). (See [2–6]). From these have come what are now known as the Rectangular Transform (RT) algorithms and the Winograd Fourier Transform (WFT) algorithms. In these algorithms, the theory deals with the reduction in the number of multiplications which, unfortunately, is usually achieved at the cost of an increase in program complexity and, sometimes, in the number of additions. For these reasons, and due to the fact that multiplication is almost as fast as addition on many general-purpose computers, there has been little use made of the powerful new computational complexity theory and the algorithms resulting from it. It is quite natural, therefore, to look about to see where RCC algorithms can be used to advantage. One is led to consider microprocessors in which multiplications are relatively slow and where the machine typically is used in dedicated applications where it is worth the effort to reduce computing time.

Previous publications have shown how RT algorithms were derived using the SCRATCHPAD system [7] for algebraic manipulation and how PL/I programs were developed which order the operations so as to reduce the number of required additions and automatically generate PL/I program statements.

The Research Signal Processor

The RSP was designed particularly for digital signal processing applications where long repetitive calculations must be performed at high speeds. Typically, such calculations involve matrix operations, moving averages, or convolution calculations, and Discrete Fourier Transform (DFT) computations for spectral analysis or correlations. The RSP machine operations are designed to make the calculation of sums of products extremely fast.

The fact that the RCC algorithms are more complicated and that the RSP must be programmed in machine language with timing highly dependent on instruction sequencing would, at first sight, discourage one from using the RCC algorithms. However, the software support described in another paper in this issue by Davies and Ris [8] provides an excellent assembler, which relieves the programmer almost entirely from pipelining considerations, and a simulator with useful debugging facilities.

Since the work to be described in this paper was done, an improved architecture for the RSP, described elsewhere in this issue by Mintzer and Peled [1], was developed for implementation by the IBM Federal Systems Division. A few of the improvements, including the addition of a guard bit on the front of the Y and Z registers, would permit an improvement in program efficiency. Some comments on these new features and their anticipated effect on the present calculations are made later.

The Rectangular Transform algorithm

The Rectangular Transform (RT) method has been described in detail elsewhere [9–12] with a list of RT convolution algorithms [10]. For the present purposes, only a very brief description of the RT algorithms is given.

The convolution which we consider here is defined by

$$y_j = \frac{1}{N}\sum_{n=0}^{N-1} h_{j-n}x_n, \tag{1}$$

where $j = 0, 1, \cdots, N-1$ and where N is the length of the data sequence, it being understood that all indices are to be taken mod N. Thus, the convolution is circular; *i.e.*, when the index $j-n$ of h_{j-n} is negative, it is replaced by $N+j-n$. An RT algorithm for computing the convolution (1) may be written

$$H_m = \sum_{n=0}^{N-1} A_{m,n}h_n, \tag{2}$$

$$X_m = \sum_{n=0}^{N-1} B_{m,n}x_n, \tag{3}$$

$$Y_m = H_m X_m, \tag{4}$$

where $m = 0, 1, \cdots, M-1$, M being the number of multiplications. Then,

$$y_j = \frac{1}{N}\sum_{m=0}^{M-1} C_{j,m}Y_m, \tag{5}$$

where $j = 0, 1, \cdots, M-1$. Unlike DFT methods, the transform matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are rectangular rather than square and the elements of two of them are simple, being mostly zeros or small integers. One of them, which we let be $\boldsymbol{A}$, has simple rational numbers as elements. Herein lies the simplicity and efficiency of the RT methods. Algorithms of the form (2)–(5) are available or can be derived for relatively small sequence lengths.

For large N-values one can perform a mapping of the arrays into multidimensional arrays and do convolutions in each of the dimensions using the algorithms discussed above. Agarwal and Burrus [9] first suggested this idea and showed that this would reduce the number of multiplications. However, the cyclic property of the convolution algorithms made it necessary to pad out the array with zeros when using their method. Ref. [10] shows a different mapping of the indices which avoids the necessity for padding with zeros. For this, however, one must require that the factors of N be mutually prime. Assume, for the sake of the following discussion, that we use three factors of N and let

$$N = r_1 r_2 r_3,$$

and let the mapping of the single index j onto the triple of indices j_1, j_2, and j_3 be defined by the equation

$$j = j_1\tau_1 + j_2\tau_2 + j_3\tau_3 \bmod N, \tag{6}$$

where $\tau_\nu = N/r_\nu$. If we define the same mapping for the index n, the convolution (1) can be written as a three-dimensional convolution

$$y_{j_1,j_2,j_3} = \frac{1}{N}\sum_{j_1=0}^{r_1-1}\sum_{j_2=0}^{r_2-1}\sum_{j_3=0}^{r_3-1} h_{j_1-n_1,j_2-n_2,j_3-n_3}\, x_{n_1,n_2,n_3}. \tag{7}$$

After mapping the one-dimensional arrays h_n and x_n into the three-dimensional arrays h_{n_1,n_2,n_3} and x_{n_1,n_2,n_3}, one may operate with algorithms of the form (2)–(5) on each of the three dimensions in turn. This may be described in operator notation, with subscripts on the operators $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ corresponding to the operations in (2), (3), and (5) for the respective sequence lengths r_ν, $\nu = 1, 2$, and 3 as follows:

$$\boldsymbol{H} = \boldsymbol{A}_3\boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{h}, \tag{8}$$

$$\boldsymbol{X} = \boldsymbol{B}_3\boldsymbol{B}_2\boldsymbol{B}_1\boldsymbol{x}, \tag{9}$$

$$\boldsymbol{Y} = \boldsymbol{H} \times \boldsymbol{X}, \tag{10}$$

$$\boldsymbol{y} = N^{-1}\,\boldsymbol{C}_1\,\boldsymbol{C}_2\,\boldsymbol{C}_3\boldsymbol{Y}. \tag{11}$$

In (8) $\boldsymbol{h}$ may be thought of as the three-dimensional $r_1 \times r_2 \times r_3$ array of elements h_{j_1,j_2,j_3} resulting from the mapping of the original sequence h_j according to (6). The result of the operation, $\boldsymbol{A}_1\boldsymbol{h}$, is the three-dimensional array obtained by applying the $\boldsymbol{A}_1$-transform (2) to the first index j_1 of $\boldsymbol{h}$. It is to be noted that, since $A_{m,n}$ in (2) is a rectangular matrix with M_1 rows, M_1 being the number of multiplications for the r_1-point algorithm, the length of the result $\boldsymbol{A}_1\boldsymbol{h}$ in the first dimension is increased to M_1. $\boldsymbol{A}_2$ and $\boldsymbol{A}_3$ have a similar effect on the second and third dimensions. A similar transformation is performed on $\boldsymbol{x}$. The transformed arrays are multiplied element-by-element in (10), and in (11) the result is transformed back, one dimension at a time, to yield the three-dimensional array $\boldsymbol{y}$. This final array is permuted according to (6) into the one-dimensional array of values of the convolution (1). The total number of elements in each of the arrays $\boldsymbol{H}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$ is

$$M = M_1M_2M_3, \tag{12}$$

which is the total number of multiplications required by the algorithm, where M_ν is the number of multiplications required for the r_ν-point algorithms.

Table 1 Number of multiplications and additions for convolution.

	$N =$	*20*	*60*	*180*
RT	Multiplications	50	200	950
	Additions	230	1120	6990
	Totals	280	1220	7940
FFT	Multiplications	152	928	1808
	Additions	356	1544	7028
	Totals	508	2472	8836
Direct	Multiplications	400	3600	32400
	Additions	380	3540	32220
	Totals	780	7140	64620

Table 2 Timing for convolution on the IBM 370/168 VM (in milliseconds).

Method	$N =$	*20*	*60*	*180*
RT		0.5	1.4	7.5
FFT		1.9	4.2	11.3
370 Assembly Language		0.7	5.2	44.6
Direct PL/I		3.1	26.1	234.3

It is seen that, by using the algorithms for the factors of N in this manner, one can construct a program by first writing convolution algorithms for each of the factors of N, r_ν, $\nu = 1$, 2, 3, and then inserting each of them in a program with loops for repeating the calculation for all values of the other two indices. As described in [10], the number of additions depends upon the ordering of the factors, because each application of the operators $\boldsymbol{A}_\nu$ and $\boldsymbol{B}_\nu$ enlarges the array. It is also shown in [10] that if A_ν, $\nu = 1$, 2, 3, is the number of additions required for the r_ν-point algorithm, then the whole calculation should take

$$A_{\text{tot}} = A_1 r_2 r_3 + M_1 A_2 r_3 + M_1 M_2 A_3 \tag{13}$$

additions. One can derive from this the rule that the factors should be placed in the order with increasing values of the quotient

$$T(r_\nu) = \frac{M_\nu - r_\nu}{A_\nu}. \tag{14}$$

It can be seen here that reducing one M_ν reduces the total number of multiplications, (12), by the factor by which M_ν is reduced. On the other hand, a change in the number of additions in one of the algorithms makes a less significant change in the total number of operations since it appears in only one of the terms of the sum in (13). This is the reason that in the multidimensional method one reduces computation by reducing the number of multiplications even at the expense of increasing the number of additions in the short convolutions. Therefore, as we shall see, the multidimensional rectangular transform method yields faster algorithms even in machines where multiplication is as fast as addition.

Results on the IBM System/370 Model 168

For the sake of comparison, we first cite some results obtained by timing the RT algorithms on a general-purpose computer. Convolution programs for $N = 20 = 4 \times 5$, $N = 60 = 4 \times 3 \times 5$, and $N = 180 = 4 \times 9 \times 5$ were written in the PL/I programming language and run on the IBM System/370 Model 168 in a time-sharing system. The factors were put in the optimal order according to the size of $T(r_\nu)$ in (14) to yield the numbers of operations listed in Table 1. The timings for the calculation of the cyclic convolution on the IBM System/370 Model 168 with the RT method and the FFT method, in floating-point arithmetic, are given in Table 2 in milliseconds. Here, and in what follows, we consider a situation where a single set of weights h_n is to be convolved with many x_n sequences. Therefore, the transform of $\boldsymbol{h}$ is pre-computed and used repeatedly so that we do not include the time for the transforms of $\boldsymbol{h}$. For all three values of N with the RT method, the permutations (6) were done without loops, *i.e.*, by stringing out the list of load and store operations. In addition, the programs were written so that the PL/I pre-processor sets up program parameters, producing very efficient object programs, each for a specific value of N. The FFT program used was an early version of Singleton's FORTRAN program, now available in the IEEE Digital Signal Processing Program Book [13]. For comparison, the timing for a very efficient machine-language program by the direct method with N^2 multiplications and $N(N - 1)$ additions is given. The last line of Table 2 gives the time taken by a simple PL/I program with two "DO" loops to show how much can be gained by the various degrees of programming effort.

The results in Table 2, of course, show an enormous improvement, *i.e.*, by a factor of 21 for $N = 180$, in going from a PL/I program using the direct method to an FFT method. Then, going from the FFT method to the RT method yields an improvement by the factors 4, 3, and 1.5 for $N = 20$, 60, and 180, respectively. In most numerical calculations, one is not very enthusiastic about the last-mentioned increase in the speed of calculation. In fact, one finds changes of this magnitude by doing the same calculation with different compilers and optimization levels within compilers. However, in digital signal processing, the factor of 1.5 in the speed of doing the 180-point convolution may produce something close to the same improvement in the total calculation, which, for speech, radar, sonar, and seismic applications may make an improvement in performance. It may even make the difference between being able to do the calculation in real time or not, or it may reduce the number of signal processing devices by this factor. Of course, for a

factorizable N less than 180, the RT method is definitely superior. A further advantage of the RT method is that, for some range of low N-values, the total program is much shorter and simpler than one using an FFT subroutine.

When the RCC algorithms were first proposed, it was anticipated that their most important applications would be in microprocessors where multiplication takes more time than addition. However, as described above, when used with multidimensional convolution methods, the number of additions as well as multiplications is reduced. Furthermore, one can see, in Tables 1 and 2, that the improvement of RT over FFT methods is more than one would predict from the numbers of multiplications and additions. This is due to the fact that the RT algorithms also require less logic and "clerical" operations.

Results with the RSP

A meaningful comparison of the advantages of one algorithm over another is complicated by the fact that there are large tradeoffs in speed *versus* program complexity, program size, and the number of constants or tables which one must store. One can also have variations in the size of the programs depending upon the number of bits of accuracy which must be preserved.

For the present RSP programs, small modules were written for each of the factors of N. These were put into programs for $N = 9$, 12, 20, 36, 60, and 180. In all cases, the calling program, written in PL/I, computed the transform of h, and the number of cycles given are for the RSP subroutine which did the rest of the calculation. The permutation of input and output data was performed by stringing out the load and store operations. A permutation routine with three nested loops, taking 55 instructions, including NOPs generated by the assembler, was written and tried but not used for the timing given here. If used with the 180-point transform, for example, it would have taken 38 cycles per output point *versus* 4 for the strung-out routine, while the number of instructions would have been 55 *versus* 720.

Table 3 gives the number of RSP cycles per output point counted by the simulator. The multiplication of the transforms was done in these programs in a loop using the built-in multiply, MPY, which takes 8 cycles, which, with the load, store, and indexing operations, results in 13 cycles for each multiplication. This would be appropriate for a situation where the hs are to be parameters of the program. If, on the other hand, the hs were to be built into the program in the form of lists of shift and add operations (coefficient store), they would have taken, on the average, about 6.5 cycles each. The figures in the third column of Table 3 are for this situation.

Table 3 Number of RSP cycles per output point for convolution.

N	*RT MPY 13 cy.*	*RT coeff. 6.5 cy.*	*Fourier transform*	*Loop MPA 13 cy.*	*Strung MPA 9 cy.*	*Coeff. store 6.5 cy.*	*Coeff. store 3.5 cy.*
9	54	40		117	81	59	32
12	53	42		156	108	78	42
20	70	54		260	180	130	70
32			277	416	288	208	112
36	85	68		468	324	234	126
60	96	74		780	540	390	150
64			314	832	576	416	224
128			353	1664	1152	832	448
180	164	130		2340	1620	1170	630
240*			139	3120	2160	1560	840
256			390	3328	2304	1664	896

*Uses a WFT algorithm. All others use a radix-2 FFT algorithm.

For a comparison with a Fourier transform method, a radix-2 FFT program supplied by Norman Brenner was used, giving the results in the fourth column except where $N = 240$. For the latter case, a program using a modified form of the WFT algorithm, provided by the members of the RSP project, was used. Before comparing with "direct" methods, it must be pointed out that the RT and Fourier methods compute cyclic convolutions, while the figures about to be discussed for the "direct" methods are for either cyclic or noncyclic convolutions. Since the former methods may require padding out with zeros when noncyclic convolutions are wanted, one may have to double the value of N being used in the comparison. We now consider four different ways of computing a convolution from the defining formula. The fifth column of Table 3 is for a loop of multiply and add (MPYA) instructions taking 13 cycles per multiplication. In the sixth column, it is assumed that the loop is unstrung. If coefficient store is used with random numbers, one should get the results listed in the seventh column, while for the usual impulse-response functions, which trail off exponentially, it would really take only about 3.5 cycles per multiplication, giving the estimates in the last column.

A comparison of the RT method with the FFT method shows it to be from 3.5 to 2 times as fast for the range $N = 20$ to $N = 180$. The Fourier transform method using the WFT, however, is about the same as an extrapolated value of the better of the two RT methods. The value $N = 240$ is, however, a particularly good one for the WFT and such superior performance cannot be expected for many other values of N. If we go now to a comparison with the last four columns of Table 3, we should first observe that the last column gives timing for a program for which the designers of the RSP built the machine for highest performance. For this column, we see a cross-over, where RT and Fourier transform method are better only for higher N-values. The RT

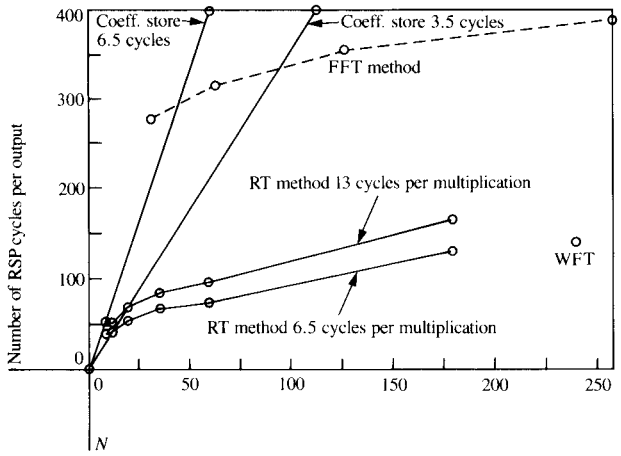

Figure 1 Number of RSP cycles per output point for cyclic convolution by using coefficient store, the FFT method, the WFT method (circle), and the RT method.

Table 4 Comparison of RSP storage* requirements for RT and FFT programs.

Method	*N*	*Instructions*	*NOPs*	*Tot. inst.*	*Data*	*Coeff.*
RT	9	237	7	244	110	0
	20	226	2	228	158	0
	60	438	9	447	608	0
	180	1133	10	1143	2912	0
FFT	32	438	68	502	77	828
	64	438	68	502	141	828
	128	438	68	502	269	828
	256	438	68	502	525	828
WFT	240	378	19	397	321	355

*Instructions: 20 bits; data: 16 bits; coefficient store: 7 bits.

and Fourier methods are better than the other three "direct" methods, so one would have to weigh the tradeoff in program size *versus* speed in deciding which to use.

To give a graphic description of the relative timing, some of the data of Table 3 are plotted in Fig. 1, where small circles have been placed at points corresponding to data in Table 3. Smooth curves are drawn between them for the sake of comparison, although no such curves can exist.

Table 4 gives the amount of RSP storage required for the programs used for Table 3. The fifth column of Table 4 gives the total number of instructions in each program, including the NOPs listed in the fourth column. The data storage requirements listed in the sixth column include the input-output arrays as well as temporary intermediate quantities, and the coefficient storage in the last column is for the special seven-bit shift and add instructions used for doing the RSP "coefficient store" multiplications.

Comparing the RT method with the FFT radix-2 method, the former is done with a simpler program which takes fewer instructions for N less than and equal to 60. The FFT program takes 828 cells of coefficient store for the sines and cosines while the RT method takes none. The $N = 180$ RT program takes more than twice as much instruction store due to the $r = 9$ algorithm. However, the program is quite simple, with three loops for the transform of x, a small six-instruction loop for the multiplication of the transforms, and three loops for the C-transform. The input and output mappings take $4 \times 180 = 720$ instructions which, as mentioned above, could have been done with 55 instructions. This would have made the program take 34 more cycles per point, in which case the RT method would still be superior to direct methods. The large amount of data store is caused by the large (950) RTs of h and x. In the RT method, there is a different program for each N while the FFT works with any power of 2. However, the RT method is about three times as fast in this range of N values. One must remember that the RT and radix-2 FFT work for different Ns, and if N is constrained, that fact may determine the method of choice.

Scaling and accuracy

Any transform method for computing convolutions in fixed-point arithmetic will have scaling and accuracy problems, since each transform value is a weighted sum of all of the data and must be scaled down to the range allowed for the data. For maximum accuracy, one should program with down-scaling during the accumulation of sums. In the RSP instruction set, this can be done with the "transfer to register Z" instruction followed by "shift and add" instructions since the accumulator contains guard bits at its low-order end. One can do the calculation faster by using the "add memory to Z" operations. However, this requires the pre-scaling of data, with consequent loss of significant bits. Scaling is particularly convenient in the radix-2 FFT algorithm since it can be done by scaling down by the factor 2 at each stage, or "butterfly." In the RT algorithm, there is a greater growth of numbers during the transforms, and a little more scaling is often required.

Given sufficiently large registers, the RT method can be made to compute convolutions of sequences of integers, giving exact integer results. However, this is not practical since it requires very large registers, and such precision is of no value when the input data are subject to error.

Two practical methods of error control are possible. The conservative approach is to scale down so that no overflow is possible, while a more radical approach is to do less scaling and permit a small probability of overflow. In the latter case, the RSP, in saturation mode, sets the arithmetic register to its highest or lowest value, depending on whether the result is positive or negative, respectively. Complete prevention of

overflow is necessary in applications where a single result has great importance, such as in setting off an alarm. However, there are many cases where the output is a graph or an audible tone in which an occasional error would not be noticed.

The RT programs were run with random number data evenly distributed in the range -1 to $+1$. Starting with the fact that sums of random numbers quickly assume a Gaussian distribution, it can be shown that the probability of overflow can be made small with somewhat less than full scaling. For example, the row of the $\boldsymbol{B}$-matrix for the nine-point RT with the largest sum of magnitudes gives

$$b_7 = (1\ 1\ 1\ 1\ 1\ 1\ 2\ 2\ 2)x. \tag{15}$$

If the elements of $\boldsymbol{x}$ are in the range $(-\rho, \rho)$, the maximum magnitude of b_7 is 12ρ, while its variance is

$$\sigma_7 = \sum_{i=0}^{8} b_{7,i}^2\, \sigma_x = \sqrt{18}\sigma_x = \sqrt{6}\rho. \tag{16}$$

Assuming a Gaussian distribution for the elements of $\boldsymbol{x}$, one obtains the probabilities of b_7 being in the range $(-1, 1)$ for the various scale factors ρ (see Table 5). Since the maximum magnitude of b_7 is 12ρ, one would have to use $\rho = 1/16$ for full scaling. However, it is seen that using $\rho = 1/8$ would produce an error, in this worst case, only once in a thousand times.

In the 180-point convolution, the PL/I program in the host computer computed and scaled the RT of $\boldsymbol{h}$ so that the maximum scaled h_n was 1. Then, various scaling parameters were used at the start of each stage, with the results given in Table 6. The columns headed by S_{x4}, S_{x9}, and S_{x5} give the scale divisors for the $r = 4$, 9, and 5 algorithms, respectively. It can be seen how the error and its variance are directly proportional to the product of the scale divisors except for the data on the last line. The last line was put in to show how the lack of scaling at the first stage increases the chances of overflow. Obviously, reducing the divisors at later rather than earlier stages reduces the chances of overflow. The next to the last line has a smaller overall scale divisor than the last line but has apparently not suffered any serious overflow problem. The variance may be compared with the variance of $\sigma = 0.025$ for the 180-point convolution, showing that for the best case in Table 6 one gets about nine significant bits. These empirical results should give some incentive to derive some rigorous formulas for optimal internal scaling for the RT, the FFT, and the WFT for general signal processing.

Suitability of the RSP instruction set

The RT algorithms, unlike the FFT and WFT algorithms, would not benefit very much by the availability of more than the one index register in the RSP. The only situation where more index registers would be of use would be where one wanted to save some instruction store and do the permutation in nested loops, and this would not be a very great saving. Due to the strung-out nature of the sequence of additions, almost no NOPs were needed, so that RT algorithms work well with the pipeline architecture. In order to avoid prescaling and retain signficant bits, it was necessary to add numbers by loading the Z register and shifting while adding. If a leading guard bit were present, as it is on the new version of the RSP, it would have been possible to add first and then shift, saving roughly one out of four operations in the addition portions of the program.

Table 5 Probabilities that b_7 will not overflow as a function of the scale parameter ρ.

ρ	$P(-1 < x < 1)$
1	0.316
1/2	0.585
1/4	0.897
1/8	0.999
1/16	1.000

Table 6 Maximum error and RMS error for various scaling divisors used with the 180-point RT algorithm.

S_{x4}	S_{x9}	S_{x5}	S_x	*Maximum error*	*RMS error*
4	16	8	512	0.01714	0.00324
4	16	4	256	0.00817	0.00160
4	8	4	128	0.00417	0.00080
4	4	4	64	0.00209	0.00042
4	4	2	32	0.00106	0.00022
2	8	4	64	0.00793	0.00310

Conclusions

Results show that when speed of processing is important, it may be worthwhile doing convolutions by the RT method instead of the FFT method for sequence lengths of moderate size. Just what "moderate" is can be judged from the execution times listed in Table 3 and graphically illustrated in Fig. 1. Since these are very roughly proportional to the numbers of operations, the tables in Ref. [10] can be used to determine which method is preferable by simply adding the numbers of multiply and add operations for the sequence lengths under consideration. The RT methods involve scaling and accuracy problems which are described and tested empirically, with the conclusion that less than full scaling, which results in a small probability of overflow error, is the most practical approach for the use of the RT algorithms on the RSP.

References

1. Fred Mintzer and Abraham Peled, "A Microprocessor for Signal Processing, the RSP," *IBM J. Res. Develop.* **26,** 413–423 (1982, this issue).
2. S. Winograd, "On Multiplication of Polynomials Modulo a Polynomial," *Research Report RC-6791,* IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1977; also, *SIAM J. Computing* **9,** 225–229 (1980).

   S. Winograd, "On Multiplication in Algebraic Extension Fields," *Research Report RC-6642,* IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1977; also, *J. Theor. Computer Sci.* **8,** 359–377 (1979).

   S. Winograd, "Some Remarks on the Effect of a Field of Constants on the Number of Multiplications," *Proceedings of the Foundations of Computer Science Symposium,* Berkeley, CA, Oct. 13–15, 1975.

   S. Winograd, "Some Bilinear Forms Whose Multiplicative Complexity Depends on the Field of Constants," (extended version), *Research Report RC-5669,* IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1976; also, *Math. Syst. Theory* **10,** 169–180 (1977).
3. S. Winograd, "On Computing the Discrete Fourier Transform," *Math. Comp.* **32,** 175–199 (1978).
4. H. F. Silverman, "An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA)," *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-25,** 152–164 (1977).
5. D. P. Kolba and T. W. Parks, "A Prime Factor FFT Algorithm Using High-Speed Convolution," *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-25,** 281–294 (1977).
6. J. H. McClellan and C. Rader, *Number Theory in Digital Signal Processing,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.
7. J. H. Griesmer, R. D. Jenks, and D. Y. Y. Yun, "SCRATCHPAD User's Manual," *Research Report RA-70,* IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1975; also, *SCRATCHPAD Technical Newsletter No. 1,* Nov. 15, 1975. See also J. H. Griesmer and R. D. Jenks, "Experience with an On-Line Symbolic Mathematics System," *Proceedings of the ONLINE 72 Conference,* Vol. 1, Brunel University, Uxbridge, Middlesex, England, Sept. 4–7, 1972, pp. 457–476.
8. Ken Davies and Fred Ris, "Real-Time Signal Processor Software Support," *IBM J. Res. Develop.* **26,** 431–439 (1982, this issue).
9. R. C. Agarwal and C. S. Burrus, "Fast One-Dimensional Digital Convolution by Multidimensional Techniques," *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-22,** 1–10 (1974).
10. R. C. Agarwal and J. W. Cooley, "New Algorithms for Digital Convolution," *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-25,** 392–410 (1977); also, *Research Report RC-6446,* IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1977; also, "New Algorithms for Digital Convolution," *Addendum, Proceedings of 1977 International Conference on Acoustics, Speech, and Signal Processing,* Hartford, CT, 1977, p. 360.
11. N. S. Reddy and V. U. Reddy, "High-Speed Computation of Autocorrelation Using Rectangular Transforms," *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-28,** 481–483 (1980).

    N. S. Reddy and V. U. Reddy, "Complex Rectangular Transforms for Digital Convolution," *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-28,** 592–595 (1980).
12. J. W. Cooley, "Some Applications of Computational Complexity Theory to Digital Signal Processing," *Proceedings, 1981 Joint Automatic Control Conference,* University of Virginia, Charlottesville, VA, June 17–19, 1981; also, *Research Report RC-8805,* IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1981.
13. *Programs for Digital Signal Processing,* Digital Signal Processing Committee, IEEE ASSP Society, IEEE Press, 345 E. 47 St., New York, NY, 1979.

Received September 28, 1981; revised February 17, 1982

The author is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.