ADAPTIVE BEAMFORMING USING RNS ARITHMETIC

Barry J Kirsch[1] and Peter R Turner[2]
1. Naval Air Warfare Center - Aircraft Division, Code 5051, Warminster PA 18974
2. Mathematics Department, U S Naval Academy, Annapolis MD 21402

Abstract

This paper is concerned with the solution of the adaptive beamforming problem using an algorithm-architecture-arithmetic combination which has the potential for solution of the problem on a small platform which might be suitable for use on aircraft or sonobuoys. The arithmetic used is the RNS system implemented on an array of processors which can be reassigned as the algorithm proceeds. The underlying algorithm is a modified Gauss elimination. The (non-RNS) division operations are eliminated in favor of some scaling and the adaptive use of the processor array to accommodate the growth in dynamic range.

Key Words RNS arithmetic, Gauss elimination

1. Introduction

If directional interference impinges an antenna array, that interference will be attenuated depending on the direction. Appropriately chosen amplitude and phase weighting of the antenna elements steer the nulls of the resulting beam pattern in the direction of the interference. The problem has been extensively studied in [5], for example. This paper describes a particular algorithm using novel processor implementations.

An array of N antenna elements are sampled at time k to form a complex snapshot vector $\vec{x}_k$. A collection of $K>N$ of these snapshots form the $N\times K$ data matrix X. Inner products between the data vector $\vec{x}_k$ and weights $\vec{w}$ yield the scalar outputs y_k. The problem is to determine the complex weights $w_0, w_1, \ldots, w_{N-1}$ that optimize the response $\vec{y}$. It is called *adaptive beamforming* (ABF) when the weights are adjusted continually.

In one formulation the weights are obtained by solving the system $R\vec{w} = \vec{s}$ where $R = XX^H/K$ is the estimated *covariance matrix*; $\vec{s}$ can be either the *steering vector* or the *cross-correlation* vector.

The concern of this paper is with obtaining solutions *quickly* on a *physically small* processing unit for operation on platforms such as aircraft or sonobuoys. Speed of numerical processing is the reason for choosing RNS arithmetic and dictates that non-RNS operations be kept to an absolute minimum; this in turn places constraints on the algorithm. The algorithm-architecture combination proposed here is based on using Gauss elimination to solve the above system.

Divisions are avoided by allowing substantial growth in the dynamic range. This is achieved by the adaptive use of an array of RNS processors and some scaling to reduce the growth of matrix elements.

In the remainder of this section, we summarize briefly the relevant aspects of residue number systems, and its extensions to complex RNS arithmetic.

Section 2 is concerned with the modified Gauss elimination algorithm and with the basic philosophy of the proposed algorithm. In Section 3, several of the subproblems and their associated difficulties are considered. These center on the questions of growth of the matrix elements and the use of adaptive RNS-base extension and scaling to handle the growth.

Section 4 brings the ideas together in a detailed description of the overall elimination algorithm. In Section 5, the back substitution phase is described.

Residue Number System (RNS) arithmetic is an exact integer arithmetic which is naturally parallel and carry-free. Due to this parallelism, RNS addition and multiplication is faster than conventional integer arithmetic for comparable dynamic range. Because of the restriction to the integers, it is often necessary to convert between RNS and standard binary for operations such as division, square roots and comparison that are not easily handled in RNS.

Signal processing tasks such as FIR's and DFT's are multiply-accumulate (MAC) intensive; RNS is ideal for these operations. More complicated algorithms such as adaptive processing, involve more non-RNS operations. The conversion of RNS numbers to binary is expensive; if enough conversions must be made, the advantages of RNS are lost. Much of this paper is concerned with how we can eliminate these operations in ABF.

For an extensive introduction to RNS see [9]. Integers are mapped to an L-tuple of residues by reducing the integer mod p_i ($1\le i\le L$) where the *moduli* p_i are relatively prime. The dynamic range, M, of the

system is the product of the moduli: $M = \prod_{i=1}^{L} p_i$ Arithmetic operations are performed on the respective elements in the L-tuples. Integers X and Y are mapped to $(x_1, x_2, ...,x_L)$ and $(y_1, y_2, ...y_L)$ where $x_i = X \bmod p_i$, $y_i = Y \bmod p_i$; addition and multiplication are then performed by componentwise modular arithmetic:

$$X + Y = \left(\langle x_1 \oplus y_1 \rangle_{p_1}, \langle x_2 \oplus y_2 \rangle_{p_2}, \cdots, \langle x_L \oplus y_L \rangle_{p_L}\right)$$
$$X \times Y = \left(\langle x_1 \otimes y_1 \rangle_{p_1}, \langle x_2 \otimes y_2 \rangle_{p_2}, \cdots, \langle x_L \otimes y_L \rangle_{p_L}\right)$$

where $\langle a \oplus b \rangle_p$, $\langle a \otimes b \rangle_p$ denote arithmetic mod p. The resulting L-tuple is the RNS representation of the sum or product which can, if desired, be converted to binary using the Chinese Remainder Theorem (CRT).

An extension of RNS is Quadratic RNS (QRNS) [1] which allows complex arithmetic using pairs of real integers. A complex integer $(a+jb)$ is mapped to a pair of real integers (z,z^*). Given a prime $p = 4k+1$ for some $k \in \mathbf{Z}$, a *Gaussian prime*, then the congruence $x^2 = -1 \bmod p$ has two solutions in the field $\mathbf{Z}_p$ that are multiplicative and additive inverses of one another. We denote them by $\hat{j}$, $\hat{j}^{-1}$ and define a mapping from the complex integers mod p, $\mathbf{Z}_p[j]$ into $\mathbf{Z}_p \times \mathbf{Z}_p$ by

$$\theta(a+jb) = (z, z^*)$$

where $z = (a+\hat{j}b) \bmod p$, $z^* = (a-\hat{j}b) \bmod p$. The (addition-preserving) inverse mapping is given by

$$\theta^{-1}(z, z^*) = \langle 2^{-1}(z+z^*) \rangle_p + j\langle 2^{-1}\hat{j}^{-1}(z-z^*) \rangle_p$$

Thus, if $\theta(a+jb) = (z, z^*)$, $\theta(c+jd) = (w, w^*)$ then, with all additions mod p, $\theta((a+c)+j(b+d)) = (z+w, z^*+w^*)$

The Gaussian primes requiring up to 8 bits are 5, 13, 17, 29, 37, 41, 53, 61, 73, 89, 97, 101, 109, 113, 137, 149, 157, 173, 181, 193, 197, 229, 233, 241.

In conventional arithmetic, complex multiplication requires four real multiplies and two real adds. A further extension [1] to QRNS is the Galois Enhanced QRNS (GEQRNS) which reduces this to just two real RNS-adds. To achieve this, map the pair (z, z^*) to their logarithms, (e_z, e_{z^*}) with respect to some generator α of $\mathbf{Z}_p$. That is, $\{\alpha^i \mid i=0,1,2,...p-2\} = \mathbf{Z}_p \backslash 0$. The integer z is therefore equivalent to $\langle \alpha^{e_z} \rangle_p$ and can be uniquely represented by this exponent. These logarithms may be added mod $p-1$ to perform multiplications: $\langle \alpha^i \alpha^j \rangle_p = \langle \alpha^{\langle i+j \rangle_{p-1}} \rangle_p$. Hence, a complex multiply requires just 2 real adds. Zero multipliers are handled as a special case.

A practical RNS system does not have infinite precision. There is a dynamic range limitation, just as there is for a conventional processor. Algorithms must be designed to control the growth of intermediate results. Periodic scaling is necessary. This scaling can not be done directly in RNS. It can be achieved by converting back to the integers and then dividing. Conversion, through CRT or Mixed Radix Conversion (MRC) algorithms, or the core function [9], carries overhead that may negate the advantages of the RNS. Therefore, we want to stay in RNS as long as possible, being careful not to overflow the dynamic range.

An analysis of the potential growth can determine when scaling is required. More frequent or earlier scaling increases the loss of precision. We therefore scale occasionally during the computation rather than prescaling the data to keep the whole process in range.

Another possibility is scaling using the L-CRT. The L-CRT operation can be partitioned into 4 stages [3], [4] which are amenable to pipelining. If a continuous stream of data is to be converted from RNS to binary, the effective conversion rate is one conversion every clock cycle after the pipeline latency delay.

The L-CRT is computed by factoring M into a real scale factor V and an integer $M' = 2^k$ such that $M = VM'$, and $0 < M' < M$. The L-CRT is given by

$$X_S = \left\langle \sum_{i=1}^{L} \left\lfloor m_i \langle m_i^{-1} x_i \rangle_{p_i} / V \right\rfloor \right\rangle_{M'}$$

where $\lfloor \bullet \rfloor$ denotes the integer-part or floor function and $m_i = \prod_{j=1, j \neq i}^{L} p_j$. The L-CRT is a residue-to-binary conversion that scales by V with error bounded by $0 \leq |X/V - X_S| < L$ which is small since $L \ll M$.

2. Modified Gauss elimination

The primary reason for selecting Gauss elimination for solving the linear system is that it is a simple algorithm using fewer arithmetic operations than other options. In ABF we face "one-off" problems so that storing the matrix factors is not important. Changes in the relative directions or strengths of the required signal and jammer result in new systems of equations.

Gauss elimination demands relatively few non-RNS operations and the divisions can be eliminated by modification of the algorithm. Symmetric factorizations such as Cholesky, also require square-roots.

The modified algorithm discussed here uses integer arithmetic performed in RNS with some scaling and range extension to achieve a solution which is a scalar multiple of the desired solution. Only the relative magnitudes of the weights are needed.

Gauss elimination for an $n \times n$ system $A\vec{x} = \vec{b}$ can be

described as an $n-1$ step process in which, at stage i we eliminate all subdiagonal entries in column i of A. We denote the matrix at the i^{th} stage by $A^{(i)}$ (so that $A = A^{(1)}$) and its elements by $a_{jk}^{(i)}$. Similarly the components of the right-hand side at stage i will be denoted by $b_j^{(i)}$. In its simplest form the algorithm is

for i=1 to n-1

for j=i+1 to n

$$m = a_{ji}^{(i)}/a_{ii}^{(i)}, \quad a_{ji}^{(i+1)} = 0, \quad b_j^{(i+1)} = b_j^{(i)} - m b_i^{(i)}$$

for k=i+1 to n $\quad a_{jk}^{(i+1)} = a_{jk}^{(i)} - m a_{ik}^{(i)}$

To complete the solution, this elimination phase is followed by the back substitution:

$x_n = b_n^{(n)}/a_{nn}^{(n)}$

for i=n-1 down to 1

for j=i+1 to n $\quad b_i^{(i)} = b_i^{(i)} - a_{ij}^{(i)} x_j$

$x_i = b_i^{(i)}/a_{ii}^{(i)}$

For general linear systems, pivoting is necessary in Gauss elimination to reduce the effect of roundoff error. The matrix here is hermitian positive definite. Partial pivoting destroys the symmetry and, more importantly, for a positive definite system it does not improve the numerical stability of the algorithm. [11]

To maximize the efficiency of our RNS processors we want to eliminate the non-RNS divisions. At this stage, simply regard this as a requirement for solution using an integer processor.

Consider therefore one step of the elimination process. Suppose that we are eliminating in column i and consider the effect of this elimination on row $j>i$. In the conventional application of Gauss elimination, we use the multiplier $a_{ji}^{(i)}/a_{ii}^{(i)}$ which requires division.

This division can be eliminated by simply "cross-multiplying" between the two rows so that for each $k>i$

$$a_{jk}^{(i+1)} = a_{ii}^{(i)} a_{jk}^{(i)} - a_{ji}^{(i)} a_{ik}^{(i)}$$

This evidently preserves the integer nature of the matrix elements but has associated costs. The most important difficulty introduced by requiring integer arithmetic is that the matrix elements can grow rapidly.

To get an idea of the rate of growth, consider just one step of the elimination in which we are effectively dealing with a 2×2 matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$. The standard Gauss elimination yields $d-b(c/a)$ in the bottom-right whereas the integer-preserving form gives $ad-bc$ which is a times that for the standard algorithm. For the full elimination this means $a_{nn}^{(n-1)}$ becomes the determinant of the original matrix - potentially a large number.

At each stage of the elimination, it is possible that the largest element could approach twice the square of the largest element at the previous stage. In the beamforming problem, the complex integer arithmetic allows the possibility of even (slightly) faster growth. For this approach to be viable we must be able to handle a very large dynamic range in the later stages of the elimination. This can be achieved in principle by the "column-parallel, parallel-channel" approach.

The basic idea is to use an array of RNS processors allocated to the various columns of the matrix. As the elimination proceeds, fewer columns are still "active". Processors used for inactive columns can be reallocated to extend the dynamic range for the remaining columns.

For the column-parallel version of Gauss elimination denote by $\vec{a}_j^{(i)}$ that part of the j^{th} column of $A^{(i)}$ below row i. We form the multiplier vector $\vec{m}^{(i)} = \vec{a}_i^{(i)}/a_{ii}^{(i)}$ and modify subsequent columns by $\vec{a}_j^{(i)} = \vec{a}_j^{(i)} - a_{ij}^{(i)} \vec{m}^{(i)}$. This is a "vector + scalar×vector" operation. The obvious modification for integer arithmetic is

$$\vec{a}_j^{(i)} = a_{ii}^{(i)} \vec{a}_j^{(i)} - a_{ij}^{(i)} \vec{a}_i^{(i)}$$

which is a "scalar×vector + scalar×vector" for which our RNS processor can be designed.

The idea behind the adaptive parallel-channel implementation of the algorithm is that a number of parallel RNS processor channels would be used, each operating with a specific modulus. The number of channels allocated to a data item determines the dynamic range for that data. Initially the processors would be divided evenly among the matrix columns.

After each stage the number of active columns is reduced by 1 but the required dynamic range is increased. The idea is to adaptively allocate processors to columns so that the number of processors grows with the dynamic range. The idea is easily illustrated for the case of four columns using 12 RNS processor channels.

The adaptive nature of the algorithm is illustrated in Figure 1 for a hypothetical 4×4 system, the first stage uses a three-dimensional RNS representation. At the next stage, only three columns are active so a fourth modulus can be used to extend the dynamic range. Similarly for the third (and final) stage of the elimination only two columns remain active so a six-dimensional representation can be used. The extent of the range extension depends on the basis elements.

The description and figure are intended to convey the broad philosophy of the solution process not the practical detail. The number of RNS channels, the choice of basis elements and control of the growth of matrix elements are all important factors.

FIGURE 1 Schematic diagram of the adaptive dynamic range allocation of RNS processors.

Active matrix	Processor											
	1	2	3	4	5	6	7	8	9	10	11	12
	RNS basis vector and residues of matrix elements											
$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} & a_{34}^{(1)} \\ a_{41}^{(1)} & a_{42}^{(1)} & a_{43}^{(1)} & a_{44}^{(1)} \end{bmatrix}$	p_1	p_2	p_3	p_1	p_2	p_3	p_1	p_2	p_3	p_1	p_2	p_3
	$a_{11}^{(1)}$	$a_{11}^{(1)}$	$a_{11}^{(1)}$	$a_{12}^{(1)}$	$a_{12}^{(1)}$	$a_{12}^{(1)}$	$a_{13}^{(1)}$	$a_{13}^{(1)}$	$a_{13}^{(1)}$	$a_{14}^{(1)}$	$a_{14}^{(1)}$	$a_{14}^{(1)}$
	$a_{21}^{(1)}$	$a_{21}^{(1)}$	$a_{21}^{(1)}$	$a_{22}^{(1)}$	$a_{22}^{(1)}$	$a_{22}^{(1)}$	$a_{23}^{(1)}$	$a_{23}^{(1)}$	$a_{23}^{(1)}$	$a_{24}^{(1)}$	$a_{24}^{(1)}$	$a_{24}^{(1)}$
	$a_{31}^{(1)}$	$a_{31}^{(1)}$	$a_{31}^{(1)}$	$a_{32}^{(1)}$	$a_{32}^{(1)}$	$a_{32}^{(1)}$	$a_{33}^{(1)}$	$a_{33}^{(1)}$	$a_{33}^{(1)}$	$a_{34}^{(1)}$	$a_{34}^{(1)}$	$a_{34}^{(1)}$
	$a_{41}^{(1)}$	$a_{41}^{(1)}$	$a_{41}^{(1)}$	$a_{42}^{(1)}$	$a_{42}^{(1)}$	$a_{42}^{(1)}$	$a_{43}^{(1)}$	$a_{43}^{(1)}$	$a_{43}^{(1)}$	$a_{44}^{(1)}$	$a_{44}^{(1)}$	$a_{44}^{(1)}$
$\begin{bmatrix} a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ a_{32}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)} \\ a_{42}^{(2)} & a_{43}^{(2)} & a_{44}^{(2)} \end{bmatrix}$	p_1	p_2	p_3	p_4	p_1	p_2	p_3	p_4	p_1	p_2	p_3	p_4
	$a_{22}^{(2)}$	$a_{22}^{(2)}$	$a_{22}^{(2)}$	$a_{22}^{(2)}$	$a_{23}^{(2)}$	$a_{23}^{(2)}$	$a_{23}^{(2)}$	$a_{23}^{(2)}$	$a_{24}^{(2)}$	$a_{24}^{(2)}$	$a_{24}^{(2)}$	$a_{24}^{(2)}$
	$a_{32}^{(2)}$	$a_{32}^{(2)}$	$a_{32}^{(2)}$	$a_{32}^{(2)}$	$a_{33}^{(2)}$	$a_{33}^{(2)}$	$a_{33}^{(2)}$	$a_{33}^{(2)}$	$a_{34}^{(2)}$	$a_{34}^{(2)}$	$a_{34}^{(2)}$	$a_{34}^{(2)}$
	$a_{42}^{(2)}$	$a_{42}^{(2)}$	$a_{42}^{(2)}$	$a_{42}^{(2)}$	$a_{43}^{(2)}$	$a_{43}^{(2)}$	$a_{43}^{(2)}$	$a_{43}^{(2)}$	$a_{44}^{(2)}$	$a_{44}^{(2)}$	$a_{44}^{(2)}$	$a_{44}^{(2)}$
$\begin{bmatrix} a_{33}^{(3)} & a_{34}^{(3)} \\ a_{43}^{(3)} & a_{44}^{(3)} \end{bmatrix}$	p_1	p_2	p_3	p_4	p_5	p_6	p_1	p_2	p_3	p_4	p_5	p_6
	$a_{33}^{(3)}$	$a_{33}^{(3)}$	$a_{33}^{(3)}$	$a_{33}^{(3)}$	$a_{33}^{(3)}$	$a_{33}^{(3)}$	$a_{34}^{(3)}$	$a_{34}^{(3)}$	$a_{34}^{(3)}$	$a_{34}^{(3)}$	$a_{34}^{(3)}$	$a_{34}^{(3)}$
	$a_{43}^{(3)}$	$a_{43}^{(3)}$	$a_{43}^{(3)}$	$a_{43}^{(3)}$	$a_{43}^{(3)}$	$a_{43}^{(3)}$	$a_{44}^{(3)}$	$a_{44}^{(3)}$	$a_{44}^{(3)}$	$a_{44}^{(3)}$	$a_{44}^{(3)}$	$a_{44}^{(3)}$

Figure 1 should not be interpreted too literally. The apparent reprogramming of processors 4-12 for the second stage is wasteful. It is pictured that way for simplicity. In practice, processors 4-12 would be unchanged and used for the base extension to be discussed shortly. Processors 1-3 are all modified for modulus p_4. Column 2 is then processed in channels 1,4-6, column 3 in 2,7-9 and column 4 in 3,10-12. Similar changes would be made at subsequent stages.

The back substitution is discussed in Section 5.

3. Difficulties with the proposed algorithm

Several difficulties are apparent in the outlined algorithm. In this section we discuss some of these and present refinements to alleviate the problems. We discuss these for a four antenna array, so that the covariance matrix will be 4×4. It is also assumed that the elements of the covariance matrix can be uniquely represented using three 7-bit moduli. The overall array will be assumed to have 16 RNS processors. Whatever operations are performed on the matrix must be repeated for the right-hand side vector but we omit details of this.

The elements of the covariance matrix are scalar products of the data vectors. These, for a 4-antenna problem, we take to be vectors of length around 16. Scalar products relative to a fixed RNS basis are readily performed in the proposed architecture provided the dynamic range is adequate for all intermediate results.

With 16 processors subdivided into 4 groups of 4 each representing a column of the matrix, each row can be computed in one RNS-channel scalar-product time. Four such operations generate the complete matrix. Note that, at this stage, we have a 4-dimensional basis vector even though the matrix elements could be stored using just three. The additional modulus accommodates some of the dynamic range growth that will be needed.

With RNS processors that are programmable for different base moduli the same basis vector can be used for all columns. Changing base moduli requires new tables to be loaded. This must be balanced against any loss of precision entailed in controlling dynamic range.

Scaling the covariance matrix to reduce the dynamic range requirement is equivalent to coarsening the input data resolution. We consider this shortly along with the question of scaling during the elimination. If scaling is used then the choice of scale factors is important.

The biggest problem is the rate of growth of matrix elements and the consequent need for range extension. For a positive definite matrix, Wilkinson [11] establishes that there is no growth for Gauss elimination. However, if integer arithmetic without divisions is used, as we saw

in the previous section, the growth rate can be of the order of squaring the largest element at each stage.

Example

Consider 4 antennas and K=16 data vectors for a signal S=0dB at 0° and a jammer J=40dB noise at 23°.

The covariance matrix involves division of the scalar products by K. Since the relative weights are required, this division can be ignored. The resulting matrix has largest element around 2.6×10^5 or 2^{18} and so can be represented in the proposed 3-dimensional RNS form.

After one step the active matrix has largest element around 2×10^7 or 2^{24}. This growth is much less than the worst case mentioned above. A 4-dimensional RNS representation would suffice although there is capacity for a 5-dimensional basis at this stage.

The next stage produces a 2×2 active matrix with largest element 2×10^{14}. Almost worst case growth has occurred. A 7-dimensional basis would suffice. (Eight channels per column are available.) The final step yields 2.7×10^{28} so that near worst case growth has again occurred but the 16 RNS channels have this range.

The growth in this example cannot be assumed to represent the general case. Without special knowledge we must allow for worst case growth which could not be accommodated in the same array without some other growth control. We discuss periodic scaling of the active matrix shortly. First, we consider the base-extensions.

There are two aspects to the base extensions needed for the algorithm. The first is the mathematical problem of finding residues relative to new basis elements of an integer given only by its residues relative to the existing basis. This question has been discussed extensively for various special cases, [2], [7], [8], [10], for example. The present situation is almost the simplest; we wish to add one or more new moduli to the basis. In our parallel architecture, any one processor would be concerned with the addition of a single basis element. The process is described in [2] using conversion from the RNS representation to the associated mixed radix system, MRS and then computing the residue of the resulting MRS representation relative to the new modulus.

The second fundamental problem of base extension is the time-penalty for reprogramming RNS processors for new base moduli and the extension operation itself.

Consider first the operation count for base extension. There is a natural parallelism in the operations for a column. For each element, n, first compute the MRS representation $n = a_0+a_1p_1+a_2p_1p_2+\ldots$ We set $a_0 = \langle n\rangle_{p_1}$ and then $a_1 = \langle p_1^{-1}(n-a_0)\rangle_{p_2}$ which requires 2 operations in the mod p_2 processor. Then computing a_2 requires three multiply-accumulate operations in the p_3 processor and so on. The a_1 computation for the next element can be concurrent with the second stage. The total time for the MRS conversion will therefore consist of the column length C times the time for computing the final modulus plus a latency time for a single element to pass through the earlier channels.

The conversion of one entry requires i operations in the i^{th} processor except that the first processor is not needed. For an RNS basis of dimension L the total number of multiply-accumulate operations to obtain the MRS representation of the first element is therefore $L(L+1)/2-1$. The rest of the column takes $(C-1)L$ giving a total of $L(2C+L-1)/2-1$ parallel operations.

This operation count must be increased since the elements of the pivot column are also converted in each processor effectively doubling the vector length to give a final count of $L(4C+L-1)/2-1$. For the first stage, $C=4$ and $L=4$ so that the delay is 37 modular multiply-accumulate times. For the final step, $C=2$, $L=8$ and the delay for this step is 59 operation times.

For each new modulus a vector of effective length $2C$ must be processed. The computation consists of $L-1$ operations. For the same two stages as above this entails 24 and 28 operations respectively.

Before this last step, the processors must be reprogrammed for the new modulus. The principal cost is the loading of two look-up tables, one for QRNS and one for GEQRNS. For 8-bit moduli these consist of 256-bytes each. Assuming a transfer rate of 4 bytes/clock this operation takes 128 cycles which is longer than the RNS-MRS conversion. We may assume that the loading and conversion are concurrent.

With a throughput of 1 multiply-accumulate/cycle, the overall delay for base-extension is therefore around 150 cycles. Because of this cost some scaling is incorporated into our algorithm. However scaling has its own associated costs - both in time and accuracy.

Suppose that at some stage of the elimination, we try to preserve the actual dynamic range by scaling and that the active matrix contains elements which are close to extremes of the range. Suppose (the components of) the elements lie in $[-M,M]$. For complex arithmetic, the worst case implies components of the form $ad-bc \in [-4M^2, 4M^2]$ which demands scaling by a factor of the order of $4M$. This is comparable to scaling the elements of the matrix by $2\sqrt{M}$ in advance of the computation.

Consider the simpler situation of multiplying two 32-bit positive integers which are close to the limits of the range and scaling the product to this range. Write $a = a_1 2^{16} + a_2$, $b = b_1 2^{16} + b_2$ where $a_1, a_2, b_1, b_2 < 2^{16}$. Now the scaled product is $[a*b/2^{32}]$ whereas the comparable prescaling yields $[a/2^{16}]*[b/2^{16}] = a_1*b_1$. For a_1, b_1 close to the extremes of the range $[ab/2^{32}] < a_1b_1 + \min(b_1,a_2) + \min(a_1,b_2)$ so that the relative

difference is around $1/\max(a_1,b_1)$ which is about 2^{-16}. The effect of scaling, for numbers close to the extremes of the range, is similar to that of prescaling, roughly *halving* the resolution of the data.

Suppose we use the same dynamic range throughout. In the above example, the components of the initial matrix had a range of about 19 bits using three 7-bit moduli, so we are indeed close to the extremes. For growth that is not quite worst case, we might expect the number of bits required to double at each stage. (This rate is achieved in the later stages in the example.)

Scaling is approximately equivalent to halving the precision of the active matrix - at each stage. Three such stages are needed which is equivalent to having fewer than three bits of precision in the covariance matrix. What effect does such degradation in precision have on the solution?

The standard error analysis for Gauss elimination is not applicable since it assumes the divisions are performed. A few preliminary experiments on quantized data were conducted to see how the performance degraded with data resolution (# bits). The same data was used for each variation in resolution and the difference in the beam plots was examined.

The adapted beam patterns $B(\theta)$ were computed for varying precisions in the input data and $|B(\theta_s) - B(\theta_j)|$ was compared to the desired solution. Here θ_s, θ_j represent the signal and jammer directions respectively. The single jammer INR=30 dB so the maximum difference is about 30 dB due to thermal noise.

TABLE 1

Degradation of solution vs input data resolution

SNR=0, INR=30 dB. Values of $20 \log_{10} |B(\theta_s) - B(\theta_j)|$

	θ_j=30°			θ_j=45°	
# bits	Trial 1	Trial 2	Trial 3	Trial 1	Trial 2
16	30 dB	28 dB	30 dB	32 dB	31 dB
5	28 dB	29 dB	30 dB	33 dB	32 dB
4	26 dB	32 dB	39 dB	30 dB	34 dB
3	28 dB	31 dB	28 dB	31 dB	31 dB
2	17 dB	21 dB	20 dB	22 dB	28 dB
1	9 dB	17 dB	singular	12 dB	15 dB

Initially this looks promising; 3 or 4 bits data resolution appear to yield tolerably good accuracy. This first impression is misleading. For data vectors of length 16, data accuracy of 1 bit yields complex scalar products which require 6 bits. Thus the final row of Table 1 used matrices with approximately twice the resolution - double the *wordlength* - of the resolution suggested by the above analysis for no dynamic range growth.

From Table 1, it appears that data resolution of 4 bits yields reasonable results. This is equivalent to an effective wordlength for the final solution of about 12 or 13 bits in the covariance matrix. Of course this is not strong evidence for the adequacy of this precision - it is evidence of the inadequacy of significantly less precision. Sensitivity of the solution to precision in the weights was considered by Nitzberg [6].

Some compromise between range extension (which costs time) and scaling (which costs precision) is needed to achieve acceptable results (in both senses).

4. The elimination algorithm

In this section, we describe in some detail the algorithm for the elimination phase of the solution. Throughout the discussion consider just a four-antenna problem using K=16 data vectors. The initial matrix elements are inner products of complex 16-vectors.

To decide how much growth can be allowed and what scaling is necessary, we give consideration first to the dynamic range which could be achieved for the final stage of the elimination using 16 processors.

The largest dynamic range which can be obtained using 16 Gaussian primes of 8 or fewer bits is 3.85×10^{34} or about 115 bits. The effective range $[-1.92\times10^{34}, 1.92\times10^{34}]$ is sufficient for the example of the previous section but is not enough to allow worst case growth.

Suppose the initial covariance matrix is represented relative to RNS-basis {73, 89, 97} which has dynamic range M_1 = [-315104, 315104] corresponding to a little over 19 bits of precision. For vector length 16 both real and imaginary parts comprise 2×16 products. This allows the data to be quantized to 7 bits which appears, from Table 1, to yield satisfactory tuning of the array.

The 16-processor array would have the initial matrix stored relative to the four-dimensional basis {73, 89, 97, 101} which accommodates some of the growth for the column 1 elimination. For the next stage only three columns are involved and a five-dimensional basis can be used. The Gaussian prime 197 is added to the basis. Now, 73×89×97×101×197 = 12,539,268,473 so that the available dynamic range at this stage is M_2 = ±6,269,634,236 which corresponds to ±32.5 bits. Worst case growth from the original dynamic range requires a range of $\pm 4M_1^2 = \pm 3.97\times10^{11} \approx 63.3\times M_2$.

If the L-CRT is used for scaling the scale factor V must be chosen so that M' is a power of 2. To avoid the risk of overflow, scaling must be done before elimination. Scaling the dynamic range at this stage to $\pm2^{32}$ (choosing $M'=2^{32}$) is equivalent to scaling the matrix to $\pm2^{15}$; choosing V_1 = 315,104×2^{-15} = 9.616210938 will suffice.

For column 2, maximal growth of the dynamic range with an eight moduli is obtained by extending the RNS-basis to {73, 89, 97, 101, 197, 229, 233, 241}. This yields $M_3 = \pm 8.06\times10^{16} \approx \pm2^{56.2}$. Worst case growth from M_2

with the scaling already applied results in the range $\pm 2^{66}$. Again the scaling must be done in advance. For a generated range $\pm 2^{56}$ we must scale the results of the previous stage to the range $\pm 2^{27}$ which requires a scale factor $V_2 = 6{,}269{,}634{,}236 \times 2^{-27} = 46.71241519$.

Even the worst case growth at the final stage can be accommodated in the 16-dimensional RNS representation using the full basis above for which $M_4 = \pm 1.92 \times 10^{34}$. (This stage consists of "real × real - complex × complex conjugate" which has a smaller growth than the more general operations used earlier.)

The scaling achieved here is essentially optimal. The final dynamic range available is approximately $\pm 2^{113}$. To keep within this the previous stage must be within the range $\pm 2^{56}$ and to stay within this the largest "power of 2" dynamic range allowable for the previous step is $\pm 2^{27}$. These are the dynamic ranges achieved here.

The effect of this scaling could be achieved by quantization of the initial data. If we trace the scaling back to the original matrix it is equivalent to scaling the initial covariance matrix to $\pm 2^{13}$. This in turn is equivalent to using initial data with 5 bits resolution. Note that the scaling proposed here would result in a smaller loss of precision since the contributions of less significant bits are retained as long as possible.

We summarize the elimination phase as Algorithm 1 below. For simplicity in this description, we only refer to 16 RNS processors and include only the operations for the 4×4 matrix itself.

ALGORITHM 1 Parallel RNS forward elimination

Input 4×4 matrix A represented in RNS with basis {73, 89, 97, 101} but scaled so that

$$|a_{ij}| \le (73 \times 89 \times 97 - 1)/2 = 315104$$

Initialize The 16 processors are initialized for moduli {73,89,97,101;73,89,97,101;73,89,97,101;73,89,97,101}
Denote by p_k the prime modulus in processor k.

1. Scale A using $V = 9.616210938$, $M' = 2^{15}$ using processors $4j$-3 to $4j$ for column j.
2. Processors 1-4:
   Reinitialize for mod 197. Modulus vector is now: 197,197,197,197,73,89,97,101,73,89,97,101,73,89,97,101
   Processors 5-16:
   Compute MRS representations of matrix elements:
   Processors 5-8 a_{i1}, a_{i2}
   Processors 9-12 a_{i1}, a_{i3}
   Processors 13-16 a_{i1}, a_{i4}
3. Compute $\langle a_{ij} \rangle_{197}$ (from the MRS representations) in processors 1-4 using processor j for column j.
4. For $i,j \ge 2$, compute $\langle a_{ij}^{(2)} \rangle_{p_k} = \langle a_{ij} a_{11} - a_1 a_{i1} \rangle_{p_k}$ using Processors 2,5-8 for $j=2$
   Processors 3,9-12 for $j=3$, Processors 4,13-16 for $j=4$
5. Scale $a_{ij}^{(2)}$ by L-CRT using processors as in Step 4 with $V = 46.7124\ 1519$, $M' = 2^{27}$
6. Processors 3,4,9-16: Compute MRS representations:
   Processors 3,9-12 $a_{i2}^{(2)}$, $a_{i3}^{(2)}$,
   Processors 4,13-16 $a_{i2}^{(2)}$, $a_{i4}^{(2)}$
   Processors 1,2,5-8: Reinitialize for mod 229, 233, 241
   Modulus vector is now: 229, 229, 197, 197, 233, 233, 241, 241, 73, 89, 97, 101, 73, 89, 97, 101
7. Compute $\langle a_{ij}^{(2)} \rangle_{p_k}$ in processors 1,2,5-8 using
   Processors 1,5,7 for $a_{i2}^{(2)}$, $a_{i3}^{(2)}$,
   Processors 2,6,8 for $a_{i2}^{(2)}$, $a_{i4}^{(2)}$
8. For $i,j \ge 3$, compute $\langle a_{ij}^{(3)} \rangle_{p_k} = \langle a_{ij}^{(2)} a_{22}^{(2)} - a_{2j}^{(2)} a_{i2}^{(2)} \rangle_{p_k}$ using Processors 1,3,5,7,9-12 for $j=3$
   Processors 2,4,6,8,13-16 for $j=4$
9. Processors 1,3,5,7,9-12:
   Compute MRS representation of $a_{ij}^{(3)}$
   Processors 2,4,6,8,13-16:
   Reinitialize for mod 109,113,137,149,157,173,181,193
   Modulus vector is now: 229, 109, 197, 113, 233, 137, 241, 149, 73, 89, 97, 101, 157, 173, 181, 193
10. Processors 2,4,6,8,13-16: Compute $\langle a_{ij}^{(3)} \rangle_{p_k}$
11. Processors 1-16: Compute $\langle a_{44}^{(4)} \rangle_{p_k} = \langle a_{44}^{(3)} a_{33}^{(3)} - a_{34}^{(3)} a_{43}^{(3)} \rangle_{p_k}$

Output $\langle a_{ij}^{(i)} \rangle_{p_k}$, $j \ge i$ where

$\vec{p}_1$ = (73,89,97,101), $\vec{p}_2$ = (73,89,97,101,197),
$\vec{p}_3$ = (73,89,97,101,197,229,233,241),
$\vec{p}_4$ = (73,89,97,101,109,113,137,149,157,173,181, 193,197,229,233,241),

5. Back substitution

The apparent need for divisions at each stage of the back substitution is the source of potential difficulty. However the divisions can be eliminated entirely since only a scalar multiple of the weight vector is required.

The back substitution begins with an upper triangular system given for $i = 1, \ldots, 4$ by

$$a_{ii}^{(i)} x_i + \cdots + a_{i4}^{(i)} x_4 = b_i^{(i)}$$

We simplify subsequent notation by rewriting this as $U\vec{x} = \vec{c}$ with elements denoted by u_{ij}, $c_i^{(1)}$.

We begin by describing the algorithm without paying any attention to the problems of element-growth and scaling. The idea is to use a column-oriented (or *column-sweep*) algorithm with implicit multiplications on the left-hand side. The final row of the system represents the equation $u_{44} x_4 = c_4^{(1)}$ and we substitute this into the previous equations to get

$$u_{44}\begin{bmatrix} u_{11} & u_{12} & u_{13} & 0 \\ 0 & u_{22} & u_{23} & 0 \\ 0 & 0 & u_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} u_{44}c_1^{(1)} - u_{14}c_4^{(1)} \\ u_{44}c_2^{(1)} - u_{24}c_4^{(1)} \\ u_{44}c_3^{(1)} - u_{34}c_4^{(1)} \\ c_4^{(1)} \end{bmatrix} = \begin{bmatrix} c_1^{(2)} \\ c_2^{(2)} \\ c_3^{(2)} \\ c_4^{(2)} \end{bmatrix}$$

The arithmetic on the right hand side must be performed but the multiplication on the left is not needed since we are only interested in the relative magnitudes of the weights.

Continuing in this manner, we finally obtain

$$u_{11}u_{22}u_{33}u_{44}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} c_1^{(4)} \\ u_{11}c_2^{(4)} \\ u_{11}c_3^{(4)} \\ u_{11}c_4^{(4)} \end{bmatrix} = \begin{bmatrix} c_1^{(5)} \\ c_2^{(5)} \\ c_3^{(5)} \\ c_4^{(5)} \end{bmatrix}$$

We can now use the final right-hand side vector as the solution setting $x_i = c_i^{(5)}$ and therefore eliminate the divisions from the back substitution completely.

Clearly substantial growth on the right-hand side of this system is likely. At the beginning of the back substitution, the first row has range $\pm 2^{15}$, the second $\pm 2^{32}$, the third $\pm 2^{56}$ while the fourth has dynamic range $\pm 2^{113}$. The "cross-multiplication" operations would generate growth up to a maximum of about 222 bits.

Our algorithm has been based on the assumption of small (typically 8-bit) RNS processors which rules out any possibility of accommodating this growth in full.

During the elimination factors from similar dynamic ranges were being multiplied but that is no longer the case for the back substitution. The later rows have greater dynamic ranges, and therefore more significant bits, than earlier ones. We therefore choose to scale the fourth row. In order to retain the same 16 base moduli the right-hand side must be kept within $\pm 2^{113}$. For the first stage we must scale the fourth row to $M' = 2^{55}$ which requires $V = 5.33819343 \times 10^{17}$.

Similar scaling is used at subsequent stages. This is summarized in Table 2. Since scalings are applied only to the right-hand side, any scaling must be applied to all elements of this vector. The scale factors used for the subsequent stages are approximately 2^{34} and 2^{17}.

What is the effect of this scaling? Consider quantities $A+a$, $B+b$ where A,B have similar magnitudes as do a,b with a,b being much smaller than A,B. Then

$$\frac{A+a}{B+b} - \frac{A}{B} = \frac{aB-Ab}{B(B+b)} = O(a/B)$$

So the error in estimating $(A+a)/(B+b)$ by A/B is of the order of (the reciprocal of) the scale factor. Thus the error in the weights caused by this scaling is around 2^{-17} - the relative accuracy of the covariance matrix.

TABLE 2

Dynamic range and scaling for back substitution

i	$u_{ij}, c_i^{(1)}$	Scaled	$c_i^{(2)}$	Scaled	$c_i^{(3)}$	Scaled	$c_i^{(4)}$	$c_i^{(5)}$
1	$\pm 2^{15}$		$\pm 2^{72}$	$\pm 2^{38}$	$\pm 2^{96}$	$\pm 2^{79}$	$\pm 2^{113}$	$\pm 2^{113}$
2	$\pm 2^{32}$		$\pm 2^{89}$	$\pm 2^{55}$	$\pm 2^{113}$	$\pm 2^{96}$	$\pm 2^{96}$	$\pm 2^{113}$
3	$\pm 2^{56}$		$\pm 2^{113}$	$\pm 2^{79}$	$\pm 2^{79}$	$\pm 2^{62}$	$\pm 2^{96}$	$\pm 2^{113}$
4	$\pm 2^{113}$	$\pm 2^{55}$	$\pm 2^{55}$	$\pm 2^{21}$	$\pm 2^{79}$	$\pm 2^{62}$	$\pm 2^{96}$	$\pm 2^{113}$

6. Conclusions

In this paper we have presented a possible algorithm-arithmetic-architecture combination for the solution of the adaptive beamforming problem. The solution uses RNS arithmetic and a modified Gauss elimination algorithm. Divisions are eliminated at the expense of a small number of scaling operations and the adaptive use of an RNS processor array to accommodate some of the growth in dynamic range inherent in the integer solution of the system.

The indications from *preliminary* simulation of the RNS processors suggest that this algorithm merits further investigation with regard to speed, precision and the effects of quantization.

References

[1] A.Z.Baraniecka & G.A.Jullien, *Residue Number System Implementations of Number Theoretic Transforms in Complex Residue Rings*, IEEE Trans ASSP 28 (1980) 285-271.

[2] R.T.Gregory & D.W.Matula, *Base Conversion in Residue Number Systems* Proc ARITH 3, IEEE, Washington DC, 1975, 117-125.

[3] M.Griffin, M.Sousa & F.J.Taylor, *Efficient scaling in the residue number system*, Proc IEEE Intl. Conf. on ASSP, 1989

[4] J.Mellot, J.Smith, E.Strom & L.Smithwick, *The Gauss Machine: A GEQRNS DSP Systolic Array*, to appear

[5] R.A.Monzingo & T.W.Miller, *Introduction to Adaptive Arrays*, Wiley, 1980.

[6] R.Nitzberg, *Computational precision requirements for optimal weights in adaptive processing*, IEEE Trans AES 16(1980) 418-

[7] K.H.O'Keefe, *A note on fast base extension for Residue Number Systems with three moduli*, IEEE TC-24 (1975) 1132-3.

[8] K.H.O'Keefe & J.L.Wright, *Remarks on Base Extension for Modular Arithmetic*, IEEE TC-22 (1973) 833-835.

[9] M.A.Soderstrand, W.K.Jenkins, G.A.Jullien, & F.J.Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE, New York, 1986.

[10] N.Szabo & R.Tanaka, *Residue Arithmetic and its Application to Computer Technology*, McGraw-Hill, 1967

[11] J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, 1965.