1

VLSI Preprocessing techniques for MUD and MIMO Sphere Detection

Geoff Knagge[1], Linda Davis[2], Graeme Woodward[2], Steven R. Weller[1]

Abstract— **While it has been shown that sphere detectors for multiuser detection (MUD) and multiple-input multiple-output (MIMO) systems can be efficiently designed, a significant hindrance to their feasibility arises from the preprocessing requirements of these algorithms. This paper investigates a number of approaches that may be taken to address this issue, and evaluates their performance in fixed point and reduced precision floating point simulations to determine the most suitable approach. A very large scale integrated circuit (VLSI) implementation strategy is also proposed to show how the chosen method may be efficiently built in hardware.**

Index Terms— **Multiuser Detection, MIMO, Sphere Detector, Tree Search, VLSI, ASIC Implementation.**

I. Problem Background

Lattice decoders, such as the sphere detector, have potential to be of great use in wireless communications systems due to their ability to greatly reduce the size of the exponential search space that needs to be processed. However, for these to be useful, they must also be practical for implementation in very large scale integrated (VLSI) circuits.

The problem considered by this paper is to find an $n-$vector, $\mathbf{s}$, given an observed m-vector, $\mathbf{y}$, and a known $m \times n$ matrix of channel coefficients and coding information, $\mathbf{H}$. In addition, the system contains a noise $m-$vector $\mathbf{n}$, and the elements of the unknown $\mathbf{s}$ are known to be in a finite set of constellation points. The system model is described by

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{n}. \qquad (1)$$

For a multiple-input multiple-output (MIMO) problem, n is the number of transmitters and m is the number of receivers. For a multiuser detection (MUD) problem, n is the number of users and m is the spreading factor.

The sphere detector has potential to provide a low complexity solution to obtain near maximum likelihood (ML) results with soft outputs [1], and in previous work we have developed an optimised version that maintains near-ML results with a greatly reduced complexity [2], [3]. However, the preprocessing requirements present design challenges for hardware implementation. In particular, to arrange the lattice into a tree structure, a decomposition of the channel matrix, $\mathbf{H}$, must take place:

$$\mathbf{H}^H\mathbf{H} = \mathbf{U}^H\mathbf{U}, \qquad (2)$$

[1]: With the School of Electrical Engineering and Computer Science, The University of Newcastle, Callaghan NSW 2308, Australia, e-mail:{gknagge,steve}@ee.newcastle.edu.au and partly supported by the Australian Research Council under Linkage Grant LP0211210; [2]: With Agere Systems, Lvl 3, 11-17 Khartoum Rd, North Ryde, NSW 2113, Australia, e-mail: {graemew,lindadavis}@agere.com

where $\mathbf{U}$ is an upper triangular matrix with real and non-negative diagonal elements. This may be obtained via a Cholesky or QR decomposition, however in their standard form both of these in contain a large number of division and square root operations. For a MIMO system in a frequency selective fading environment, it is also of benefit to implement a noise whitening filter to mitigate the colouring effect of the equaliser [4].

For each search instance, it is necessary to calculate where the received signal vector, $\mathbf{y}$, lies within the lattice of possible original transmissions. This is the unconstrained ML estimate, $\hat{\mathbf{s}}$, and needs to be calculated for every $\mathbf{y}$. It is defined as a multiplication with the pseudo-inverse of the channel matrix :

$$\hat{\mathbf{s}} = (\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H\mathbf{y}. \qquad (3)$$

Since (3) involves a matrix inverse operation, it is generally not appropriate to directly implement this in hardware.

These design challenges must be met in order to find a complete implementation for the sphere detector. This paper investigates these problems, and develops an implementation strategy showing how they may be addressed. Section II describes an implementation for a feasible QR decomposition. However, this still requires some divisions and square roots; these are addressed in Section III. The final stage of preprocessing, the search centre calculation, is investigated in Section IV. Finally, an implementation is discussed in Section V and the paper concludes with a summary of our findings.

II. QR Decomposition

The QR decomposition is preferred over the Cholesky alternative for reasons of numerical stability [5], and this paper will later show that its outputs are useful for other preprocessing operations. For an $m \times n$ matrix $\mathbf{H}$, its QR decomposition is $\mathbf{H} = \mathbf{QR}$, where $\mathbf{R}$ is an $m \times n$ upper triangular matrix, $\mathbf{Q}$ is an $m \times m$ orthonormal matrix ($\mathbf{QQ}^H = \mathbf{I}$), and such a transformation exists for any matrix.

From a VLSI implementation perspective, the standard QR decomposition is very complex due to its requirement for division and square root operations. This is addressed in [5], which presents the "scaled and decoupled" QR decomposition that separates the numerators and denominators to avoid these VLSI unfriendly operations. This produces

$$\mathbf{H} = \mathbf{QR} = \mathbf{\Phi}^H\mathbf{K}^{-1}\mathbf{P}, \qquad (4)$$

where $\mathbf{R} = \mathbf{K}^{-\frac{1}{2}}\mathbf{P}$ and $\mathbf{Q}^H = \mathbf{\Phi}^H\mathbf{K}^{-\frac{1}{2}}$. The matrices $\mathbf{P}$ and $\mathbf{\Phi}$ are dimensioned the same as $\mathbf{R}$ and $\mathbf{Q}$ respectively, and $\mathbf{K}$ is a real-valued $n \times m$ diagonal matrix. To obtain $\mathbf{R}$ and $\mathbf{Q}$ from these results, some square roots and divisions are necessary, and are addressed in Section III.

This section proposes an example architecture for calculating the QR in hardware, which breaks the algorithm into three distinct segments, as outlined in Alg. II.1.

Algorithm II.1 Scaled and Decoupled QR decomposition

```
For i = 1 to min(n, m) do {
    For j = i+1 to n do {
        Part A   - Calculate Givens Rotation
        Part B   - Recalculate $\mathbf{P}(i, j+1)$ to $\mathbf{P}(i, n)$
                 - Recalculate $\mathbf{P}(j, j+1)$ to $\mathbf{P}(j, n)$
        Part C   - Recalculate $\mathbf{\Phi}(i, 1)$ to $\mathbf{\Phi}(i, m)$
                 - Recalculate $\mathbf{\Phi}(j, 1)$ to $\mathbf{\Phi}(j, m)$
    }
}
```

In the following, the implementation of each of these parts are discussed.

A. Givens Rotation Calculation

Part A effectively generates a Givens matrix, $\mathbf{G}$, which is an identity matrix with four entries, $\mathbf{G}_{i,i}$, $\mathbf{G}_{i,j}$, $\mathbf{G}_{j,i}$ and $\mathbf{G}_{j,j}$, modified to obtain the required transformations. It also allows a new value of $\mathbf{P}(i,i)$ to be directly obtained.

This part mainly consists of a series of multiplications by real numbers, and some addition and scaling operations. The basic flow of data can be seen in Fig. 1.

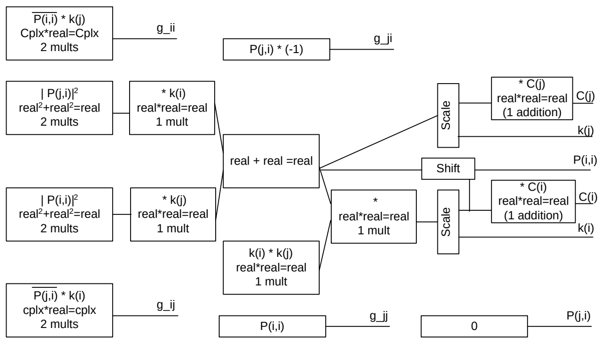
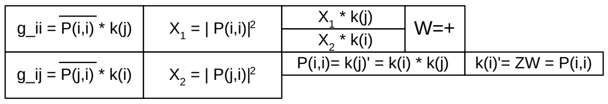


Fig. 1. Flow of data through part A of the QR decomposition algorithm.

This figure shows that there are several calculations that are dependent on each other, plus a few that are independent. These independent elements include the Givens rotation matrix, $\mathbf{G}_{xx}$, and so these may be calculated first to allow parts B and C to proceed in parallel before this part has completed.

By defining each calculation as one work unit, equivalent to an integer number of hardware clock cycles, the fastest possible throughput is limited by the number of dependent calculations. Therefore, using the work timeline provided by Fig. 2, only four work units will be needed, regardless of the size of the decomposed matrix. In terms of circuit complexity, only four real number multipliers are needed, which is the equivalent of a single complex multiplier.



g_ii = $\overline{P(i,i)}$ * k(j)
X_1 = \| P(i,i)\|²
X_1 * k(j)
X_2 * k(i)
W=+
g_ij = $\overline{P(j,i)}$ * k(i)
X_2 = \| P(j,i)\|²
P(i,i)= k(j)' = k(i) * k(j)
k(i)'= ZW = P(i,i)

Fig. 2. Flow of data through part A of the QR decomposition algorithm.

B. Rotation of $\mathbf{P}$ and $\mathbf{\Phi}$

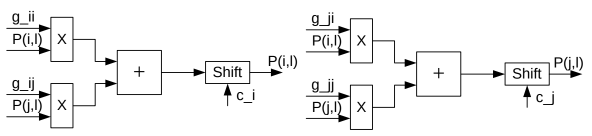


Fig. 3. Architecture for recalculation of one column of $\mathbf{P}$ or $\mathbf{\Phi}$, with l iterating from $j+1$ to n and 1 to n for each case respectively.

As seen in Fig. 3, parts B and C are quite straightforward, using the same operations of four complex multiplications and two additions per column, per iteration. With only four parallel complex multipliers, only one work unit is required for the calculation each column of $\mathbf{P}$. On each loop of the decomposition, this part must multiply the existing $\mathbf{\Phi}$ or $\mathbf{P}$ by the Givens matrix $\mathbf{G}$ that was obtained from part A. Part C is only necessary if the $\mathbf{Q}$ part of a QR decomposition is required.

C. Effect of the size of the channel matrix

By noting the independence between calculations, and using parallelism between parts, a decomposition for a 4x4 MIMO channel can be implemented in under 60 work units. However, the algorithm is easily scalable for larger matrices, but will require a longer execution time. The additional effort occurs from the need for more iterations to complete the decomposition, but each iteration only increases linearly in complexity.

The first part, of obtaining the new diagonal element and Givens matrix elements, requires no additional effort per iteration. Stages B and C will require the processing of more matrix cells, due to the wider $\mathbf{P}$ and $\mathbf{\Phi}$ matrices. A simple analysis will show that if $m = n$, then the complexity increases in $O(n^3)$.

However, it does not affect the number of multipliers required since the number of multiplications required per modified cell is unchanged. If more multipliers are available, these may be readily utilised to introduce parallelism and thus reduce the time requirements of the algorithm. An estimate of the minimum work units required for QR decomposition of various sized matrices is shown in Table I.

Dimensions	Multipliers	QR units	Total units
4×4	4	60	136
8×8	8	235	508
12×12	12	600	1216
16×16	16	1100	2324
32×32	32	4500	12036

TABLE I
PREPROCESSING WORK UNITS REQUIRED FOR DIFFERENT SIZED SYSTEMS

III. Scalar Reciprocal and Reciprocal Square Root

While the scaled and decoupled method greatly reduces the number of square roots and divisions required, it does not completely eliminate them. In particular, to obtain $\mathbf{R}$, the calculation is

$$\mathbf{R} = \mathbf{K}^{-\frac{1}{2}}\mathbf{P}, \tag{5}$$

We consider two possibilities in the following sections.

A. Newton-Raphson Method

The Newton-Raphson (NR) algorithm is an iterative method for finding the zeros of a funtion, $f(x)$, with each step calculating:

$$x_{n+1} = x_n - \frac{f\left(x_n\right)}{f'\left(x_n\right)} \tag{6}$$

Geometrically, the NR method works by finding the tangent to the curve $y = f(x)$ at the point x. The tangent will cross the x-axis at some point x_2, which becomes our new estimate. If the gradient properties of the curve are favourable, then this method will converge to the required point.

The inverse square root of a number y is found with

$$\begin{aligned} f(x) &= x^{-2} - y & (7) \\ f'(x) &= -2x^{-3} & (8) \\ x - \frac{f(x)}{f'(x)} &= \frac{3x - x^3 y}{2} & (9) \end{aligned}$$

The graphs of the functions $f(x)$ for the case $y = 5$ are presented in Figure 4, where it becomes obvious why the starting point is critical to the convergence of the Newton method.

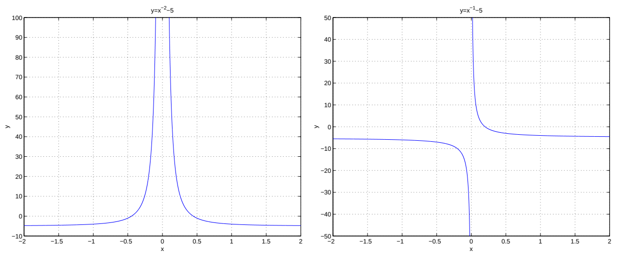


Fig. 4. Functions for which the zero needs to be found when trying to find the inverse and inverse square root of y=5.

In the case of the inverse, there is only one solution and so the search is on the positive half of the graph. However, if the search is done too far to the right, then the magnitude of the tangent's gradient will be such that it will cross the x-axis on the wrong side of the plot. From there, the result will diverge towards $-\infty$. The critical point for this is found to be $x = 0$ or $\frac{2}{y}$.

For both problems, x must lie between a pair of critical points for convergence to occur. Although the initial x could be set to a very small constant to guarantee convergence for most cases, the steepness of the gradient will cause the solution to take a long time to converge. Unless the numbers to be used are confidently known, the NR method is limited in its suitability.

B. Bisection Based Approach

The bisection algorithm assumes the known existence of a solution between two points, a and b. On each iteration, a midpoint $c = \frac{a+b}{2}$ is selected, and it is decided whether the solution is between a and c, or b and c, and either a or b is replaced with c as appropriate.

For the case of finding the inverse of a scalar x, for any candidate point y this involves calculating :

$$\text{cost} = 1 - xy \tag{10}$$

For an appropriate a and b, $cost_a$ would be positive and $cost_b$ would be negative. The midpoint, $cost_c$, would replace a if it were positive, or replace b if it were negative. As the solution converges to the correct answer, both costs converge to 0.

Similarly, finding the inverse square root of x, for any candidate point y, involves calculating :

$$\text{cost} = 1 - xy^2 \tag{11}$$

Conveniently, the computational load is quite small when applied to the binary domain. For the reciprocal square root in a floating point system, the original value is $x_m 2^{x_e}$, so the result will be of the form $y_m 2^{\lfloor \frac{x_e}{2} \rfloor}$. In a floating point system, it is known that x_m is between $1.00\ldots00_2$ and $1.11\ldots11_2$. Therefore, y_m will be between $0.10\ldots00_2$ and $1.00\ldots00_2$ for both the inverse and inverse square root operations.

The square root requires that x_e be halved, and so if this is odd, then x_m needs to be multiplied by 2 and x_e reduced by 1. However, the range of y_m is unchanged in this situation.

So, the initial case is:

$$\begin{aligned} A &= 1.000000_2 & (12) \\ B &= 0.100000_2 & (13) \\ C &= 0.110000_2 & (14) \end{aligned}$$

For the case of finding the inverse square root of 1.00101_2 (1.15625_{10}), the costs are:

$$\begin{aligned} A &: \quad 1 - xy_A^2 = 1 - 1.15625 * 1^2 = -0.15625 & (15) \\ B &: \quad 1 - xy_B^2 = 1 - 1.15625 * 0.5^2 = 0.7109 & (16) \\ C &: \quad 1 - xy_C^2 = 1 - 1.15625 * 0.75^2 = 0.3496 & (17) \end{aligned}$$

Since the C cost is positive, it will replace B and a new iteration will commence with a new $C = 0.111000_2$. For each iteration, a decision is made about an individual bit of the result. The test case C is the progressive result with a 1 added to the left as a new least significant bit. If the cost of C is positive, then the decision is kept, otherwise that bit is set to zero and the previous decision is kept. The time of the algorithm then becomes deterministic, at one iteration per bit of precision.

The calculation of costs may also be greatly simplified. Assume the result so far is A, with a cost of x. i.e.,

$$x = 1 - A^2 y \tag{18}$$

The cost of adding 0.01_2 is

$$\begin{aligned} x_{next} &= 1 - (A + 0.01_2)^2 y & (19) \\ &= x - 0.10_2 Ay - 0.001_2 y & (20) \end{aligned}$$

The multiplications by 0.10_2 and 0.001_2 are virtually free in hardware, and the multiplication Ay can be cheaply obtained in a similar manner. Therefore, the cost of each iteration is little more than a series of additions, and the result may be found quite cheaply.

A similar, but even more simplified, approach may be taken for finding the inverse of a number. In that case, the cost of adding 0.01_2 is

$$\begin{aligned} x_{next} &= 1-(A+0.01_2)y \qquad (21)\\ &= x-0.01_2 y \qquad (22)\end{aligned}$$

The bisection style approach is only useful in cases where the curve between the chosen start points is continuous with no changes in sign of the gradient. However, for the applications being considered here, that is not an issue and so the described method and optimisations are a suitable candidate. If fact, in a floating point implementation this algorithm is an ideal candidate for providing a small, efficient, and robust solution.

The architecture of these units is shown in Fig. 5 and Fig. 6. Since time is not a major constraint for these units, they can be kept small by operating on each bit of the result sequentially. Although each design contains a number of flip flops, many of these are shift registers propagating only a single “1” bit, and so consume minimal power.

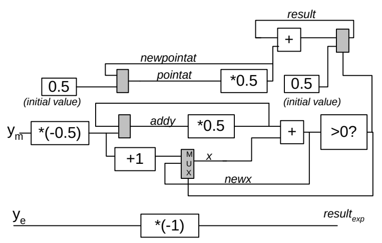


Fig. 5. Scalar Inverse Unit. Shaded boxes represent flip-flops.

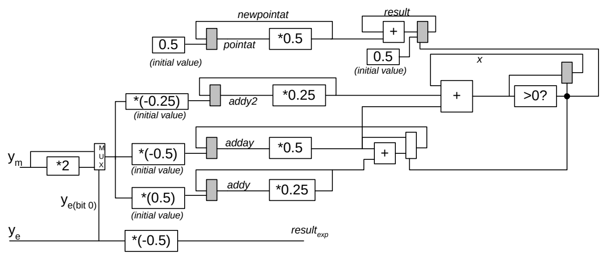


Fig. 6. Inverse Square Root Unit. Shaded boxes represent flip-flops.

IV. Search Centre Calculation

While the channel decomposition only needs to be performed once per channel update, the unconstrained ML estimate, $\hat{\mathbf{s}}$, needs to be calculated for every search operation, as defined in (3). This section evaluates each technique in terms of computational complexity and numerical stability.

A. Back substitution

There are two possible methods for finding the search centre via back substitution, depending on whether or not the matrix $\mathbf{\Phi}$ is calculated. Since $\mathbf{P}$ is in upper triangular form, $\mathbf{x}$ in (29) and $\hat{\mathbf{s}}$ in (30) can be solved by back substitution, without determining $\mathbf{\Phi}$.

$$\begin{aligned} \mathbf{H}^H\mathbf{H}\hat{\mathbf{s}} &= \mathbf{H}^H\mathbf{y} \qquad (23)\\ \left(\mathbf{\Phi}^H\mathbf{K}^{-1}\mathbf{P}\right)^H\left(\mathbf{\Phi}^H\mathbf{K}^{-1}\mathbf{P}\right)\hat{\mathbf{s}} &= \mathbf{H}^H\mathbf{y} \qquad (24)\\ \mathbf{P}^H\mathbf{K}^{-1}\mathbf{\Phi}\mathbf{\Phi}^H\mathbf{K}^{-1}\mathbf{P}\hat{\mathbf{s}} &= \mathbf{H}^H\mathbf{y} \qquad (25)\end{aligned}$$

Using the knowledge that $\mathbf{Q}$ is orthonormal,

$$\begin{aligned} \left(\mathbf{\Phi}^H\mathbf{K}^{-\frac{1}{2}}\right)^{-1} &= \left(\mathbf{\Phi}^H\mathbf{K}^{-\frac{1}{2}}\right)^H \qquad (26)\\ &= \mathbf{K}^{-\frac{1}{2}}\mathbf{\Phi}. \qquad (27)\end{aligned}$$

This allows the problem to be simplified to solving

$$\mathbf{P}^H\mathbf{K}^{-1}\mathbf{P}\hat{\mathbf{s}} = \mathbf{H}^H\mathbf{y} \qquad (28)$$

This can be solved via two back-substitution operations without use of the $\mathbf{\Phi}$ matrix. Equation (29) is solved for $\mathbf{x}$, where $\mathbf{x} = \mathbf{K}^{-1}\mathbf{P}\hat{\mathbf{s}}$, and (30) can then be solved for $\hat{\mathbf{s}}$:

$$\begin{aligned} \mathbf{P}^H\mathbf{x} &= \mathbf{H}^H\mathbf{y} \qquad (29)\\ \mathbf{P}\hat{\mathbf{s}} &= \mathbf{K}\mathbf{x}. \qquad (30)\end{aligned}$$

Back substitution, like most of the alternatives, involves divisions by each of the diagonal elements of the matrix, however these divisions can be precomputed so that only subtractions and multiplications are required. The remaining complexity for an $n \times n$ matrix consists of:

- $\frac{n(n-1)}{2}$ multiplications
- n subtractions
- n multiplications by the inverted diagonal

Additional calculations that must also be found are the results $\mathbf{H}^H\mathbf{y}$ and $\mathbf{K}\mathbf{x}$. For (29), $\mathbf{H}^H\mathbf{y}$ requires mn multiplications and $n(m-1)$ additions, and in (30) only n multiplications are required because $\mathbf{K}$ is diagonal.

In an implementation, these calculations may use a high degree of parallelism, limited only by the consequences of the time and complexity tradeoffs, as discussed in [4]. However, the amount of computations that are required for each search centre estimate remains high.

Alternatively, if $\mathbf{\Phi}$ is available, a single back-substitution may be used to obtain $\hat{\mathbf{s}}$.

$$\begin{aligned} \hat{\mathbf{s}} &= \left(\mathbf{H}^H\mathbf{H}\right)^{-1}\mathbf{H}^H\mathbf{y} \qquad (31)\\ &= \left(\mathbf{P}^H\mathbf{K}^{-1}\mathbf{\Phi}\mathbf{\Phi}^H\mathbf{K}^{-1}\mathbf{P}\right)^{-1}\mathbf{P}^H\mathbf{K}^{-1}\mathbf{\Phi}\mathbf{y} \qquad (32)\\ &= \left(\mathbf{P}^H\mathbf{K}^{-\frac{1}{2}}\mathbf{Q}\mathbf{Q}^H\mathbf{K}^{-\frac{1}{2}}\mathbf{P}\right)^{-1}\mathbf{P}^H\mathbf{K}^{-1}\mathbf{\Phi}\mathbf{y} \qquad (33)\\ &= \left(\mathbf{P}^H\mathbf{K}^{-1}\mathbf{P}\right)^{-1}\mathbf{P}^H\mathbf{K}^{-1}\mathbf{\Phi}\mathbf{y} \qquad (34)\\ &= \mathbf{P}^{-1}\mathbf{K}\mathbf{P}^{-H}\mathbf{P}^H\mathbf{K}^{-1}\mathbf{\Phi}\mathbf{y} \qquad (35)\\ &= \mathbf{P}^{-1}\mathbf{\Phi}\mathbf{y} \qquad (36)\end{aligned}$$

To avoid the need for a matrix inversion, back substitution is used to solve

$$\mathbf{P}\hat{\mathbf{s}} = \mathbf{\Phi}\mathbf{y} \tag{37}$$

The above derivation assumes the existence of the inverse $\mathbf{P}^{-1}$, which will always exist for the cases of interest.

In terms of complexity, the difference between the two back-substitution methods is a tradeoff between preprocessing and per-symbol calculations. If the calculation of $\mathbf{\Phi}$ is allowed during the QR decomposition, then the effort required to calculate $\hat{\mathbf{s}}$ is halved.

B. *Calculating* $(\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$ *with Gauss-Jordan elimination*

An obvious method for calculation of the search centre involves the direct calculation of the pseudo-inverse of the channel matrix $\mathbf{H}$, as a precomputation, followed by the multiplication by the received vector $\mathbf{y}$. However, matrix inverses are very expensive in terms of VLSI implementation.

A common method of performing a matrix inverse is via Gauss-Jordan elimination [6]. For an $m \times m$ matrix $\mathbf{H}^H\mathbf{H}$, $m^3 - m$ multiplications, m divisions, and $\frac{2m^3-3m^2+m}{2}$ additions or subtractions are required. These calculations only need to be performed once per channel update. For every symbol, it is necessary to multiply the pseudo-inverse by the received vector to obtain the search centre estimate, requiring mn multiplications.

C. *Calculating* $\mathbf{H}^{-1}$ *with Gauss-Jordan elimination*

In certain cases, the direct inverse of the channel matrix may be substituted for the pseudo-inverse. The use of this variation is limited to square channel matrices where such an inverse exists, and is well known to be less numerically stable than other methods. The computational requirements for the inverse are the same as above, but the computational advantage is that no additional matrix multiplications are required.

D. *Calculating* $(\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$ *by inverting* $\mathbf{P}$

The logical alternative to the second back-substitution method is to simply calculate equation (36) by finding $\mathbf{P}^{-1}$. While in principle an inverse is calculated, in practice only half of the effort is required because $\mathbf{P}$ is already in an upper triangular form. Direct inversions are often avoided due to concerns over numerical stability, however from the derivation, it can be seen that in this case it is no different to the back-substitution method.

E. *Calculating* $(\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$ *with Newton-Raphson method*

The Newton-Raphson method can also be extended to matrices. To find the inverse of a Hermitian matrix $\mathbf{A}$, the following iterations are performs:

$$B_{n+1} = 2B_n - B_nAB_n$$

In this case, the calculation of interest is $(\mathbf{H}^H\mathbf{H})^{-1}$ and, since $\mathbf{H}^H\mathbf{H}$ is Hermitian, the above technique can be applied. The advantage of this method is that the iterations requires no division operations. This method removes divisions from the calculations but, as with the scalar case, is dependent on choosing an appropriate starting point.

The complexity of this method is determined by the number of iterations that are performed, and the number of multiplications required can be reduced by simplifying the equation:

$$\begin{aligned}\mathbf{B}_{n+1} &= 2\mathbf{B}_n - \mathbf{B}_n\mathbf{A}\mathbf{B}_n && (38)\\ &= \mathbf{B}_n\left(2\mathbf{I} - \mathbf{A}\mathbf{B}_n\right) && (39)\end{aligned}$$

The equation and the properties of its matrices make it very similar in form to the noise whitener described in [4], and so large portions of the same datapath may be reused for this problem.

For the general case of an $m \times n$ channel matrix $\mathbf{H}$, this finds the inverse of a $n \times n$ matrix, and so requires the following operations to calculate $\hat{\mathbf{s}}$:

- The calculation of the $n \times n$ matrix. Each cell of this matrix requires m multiplications and $m-1$ additions. However, observing that the resultant matrix is Hermitian, it is only necessary to explicitly calculate the upper triangle of this matrix. Therefore, a total of $m\frac{n(n+1)}{2}$ multiplications are needed.
- The inverse can then be calculated using the iterative technique. Each multiplication of pairs of n x n matrices normally requires n^3 multiplications, but once again the Hermitian properties allow this to be simplified to $n\frac{n(n+1)}{2}$ operations.
- The pseudo-inverse is obtained by multiplying the inverse by $\mathbf{H}^H$. This requires n^2m multiplications.
- For each symbol received, the only multiplication is between the pseudo-inverse and the received vector. A total of nm^2 multiplications are required for this operation.

In summary, with i iterations, the preprocessing stages require a total of $\frac{1}{2}\left(3n^2m + mn + in^3 + in^2\right)$ multiplications on each channel update, and a further mn multiplications are needed to find $\hat{\mathbf{s}}$ for each symbol.

F. *Evaluation of alternatives*

Regardless of the computational complexity of the alternative algorithms, which are summarised in Table II, the most important characteristic is the performance and how it performs numerically. In this case, the main items of interest are

- The accuracy in limited numerical precision systems
- The behaviour of the algorithm in special situations, such as rank deficient channel matrices
- The general stability of the algorithm

For each alternative, a large number of test cases were simulated and the mean square errors (MSE) between the full and restricted precision floating point cases were measured. The median, plotted in Fig. 7, indicates that the best overall performers involve the direct inversion of $\mathbf{P}$, and back substitution with $\mathbf{\Phi}$.

The median does not take into account the effect of instabilities caused by an occasional poorly conditioned channel, and this is illustrated in Fig. 8 by plotting the number of times the MSE exceeds an arbitrary threshold of 1.

Property	Calculation of $\mathbf{P}^{-1}$	Newton-Raphson (i iterations)	Direct Calculation of $\mathbf{H}^{-1}$	Direct Calculation of $(\mathbf{H}^H\mathbf{H})^{-1}\mathbf{H}^H$	Back Substitution with $\mathbf{\Phi}$	Back Substitution without $\mathbf{\Phi}$
Computations on each Update						
Multiplications	218	104+40i	60	124	144	0
Additions	166	78+30i	42	96	90	0
Divisions	4	1	4	4	4	4
Computations on each Symbol						
Multiplications	16	16	16	16	16	40
Additions	12	12	12	12	12	24
Divisions	0	0	0	0	0	0

TABLE II
COMPARATIVE COMPLEXITIES AND ADVANTAGES FOR VARIOUS METHODS OF FINDING THE SEARCH CENTRE IN A SYSTEM WITH A 4X4 CHANNEL MATRIX.

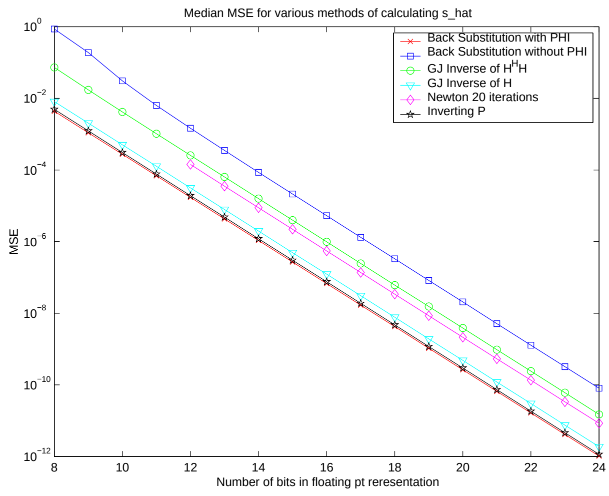


Fig. 7. Median performance of the search centre calculation techniques under restricted precision floating point constraints.

When considering both the complexity and stability together, the most favourable choice involves the direct inversion of $\mathbf{P}$. This method involves the least work for each search centre calculation, performs better and is more stable than the alternatives, and does not require significant additional precomputation effort.

G. *Floating Point vs Fixed Point*

Software simulations show that the preprocessing engine can be presented with a large range of data values, and so a fixed point implementation would require a large bit-width to achieve reasonable results. This is confirmed in Fig. 9, which indicates that approximately 6 additional bits are needed in a fixed point implementation to achieve a result comparable to the floating point simulation. Additionally, it was found that all of the trialled search centre methods have a very high instability when less than 14 bits are used in fixed point.

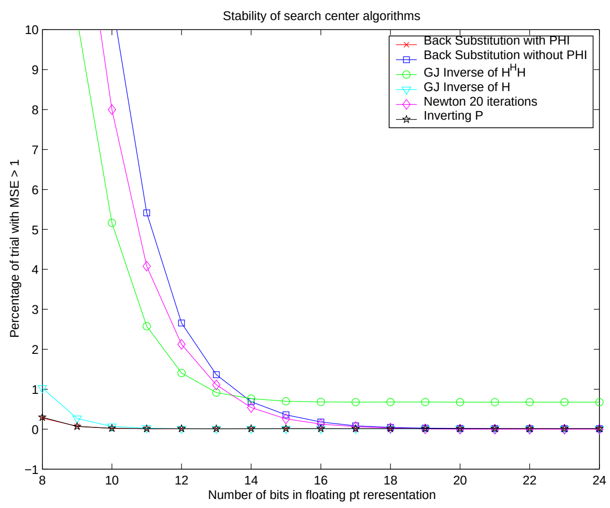


Fig. 8. Stability of the search centre calculation techniques under restricted precision floating point constraints.

In addition to reducing the number of bits needed, this also allows other optimisations to reduce power. Although the floating point design requires a five bit exponent, this does not play a part in the multiplier circuits, and so the power and size requirements can be greatly reduced.

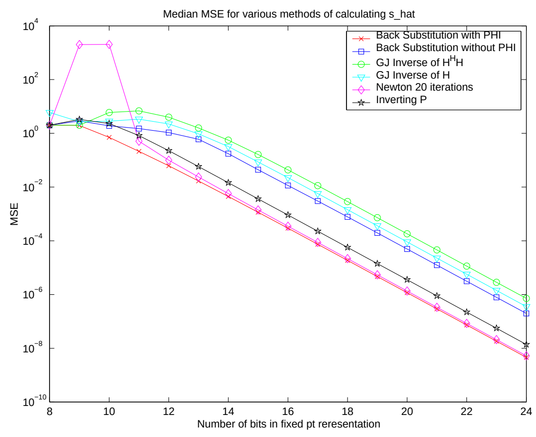


Fig. 9. Performance of the search centre calculation techniques under restricted precision fixed point contraints.

This comparison was repeated for the noise whitening algorithm presented in [4], showing that using an 8-bit floating point system achieves superior results compared to the 16 bit fixed point implementation previously proposed.

The main overhead cost in a floating point system is the need to continually rescale before any addition operation, so that all terms have the same exponent, and scale the result so that the mantissa is at least 1 and less than 2. However, a comparable cost is incurred by a flexible fixed point system, since this involves the need to check for an overflow in the result and clamp it at its extreme value where necessary. Hence, some form of shifting or scaling process is required regardless of the numerical representation used.

V. Implementation

The individual units may be combined to share hardware and form a single preprocessing engine. The interaction between components is shown in Figure 10. The "rotation calculator" contains the four real multipliers used to calculate part A of the QR algorithm and, apart from the control and memory blocks, the only other major component is the complex multiplier array. Each multiplier performs the calculation $(A+iB)+(C+iD)\times(E+iF)$, and as such may be reused to calculate parts B and C of the QR, and also the inversion and other matrix or vector multiplications.

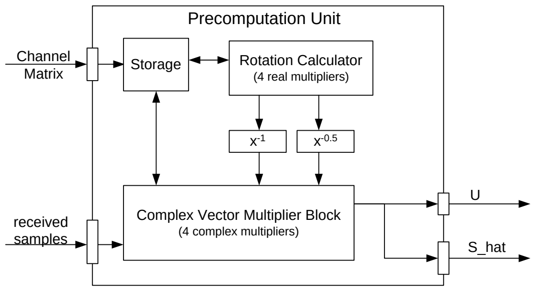


Fig. 10. Architecture of preprocessing unit for a 4x4 MIMO channel.

A bit-accurate software model of the preprocessor was used with our optimised tree search algorithm [2] to determine the effect of varying the bit rate. These results, shown in Fig. 11 and Fig. 12, reveal that as little as 10 significant bits are necessary to obtain near-optimal performance. An array of 16 pre-optimised complex multipliers, with a bit width of 12 and a 6 bit exponent, was synthesised in a 0.13μm process at 200MHz to an area of 1.5mm^2. This indicates that the entire unit may be built to an acceptable size in an optimised design.

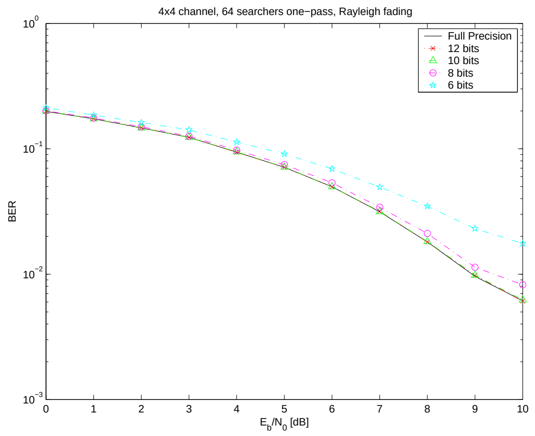


Fig. 11. Performance of preprocessor under various bit widths for a 4x4 MIMO channel.

Table I, presented in section II, shows estimates of the total work units required for various sized channels, showing that the required processing time is not excessive.

VI. Conclusion

The sphere search is a promising technique for finding near optimal results for combinatorial search problems, such as

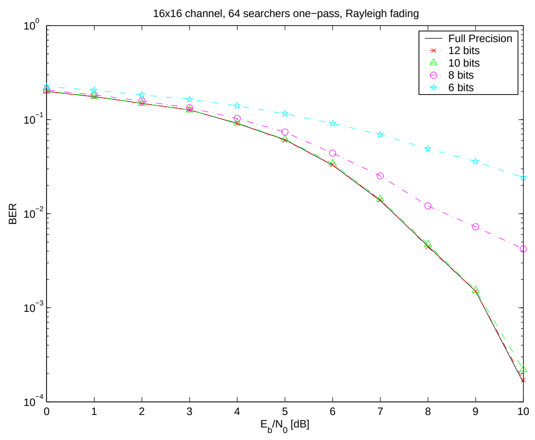


Fig. 12. Performance of preprocessor under various bit widths for a 16x16 MIMO channel.

those required for MUD and MIMO detectors. However, until now, its feasibility for VLSI has been restricted by a large overhead cost in the preprocessing required.

This paper has shown how the scaled and decoupled QR algorithm can be optimised and implemented with low complexity. Complexity and performance comparisons have been presented between various methods of calculating the search centre, and it has been found that the most appropriate implementation involves a floating point approach that reuses results from the QR decomposition. These findings have been combined into a single VLSI design that would be of a feasible size and operates at a real-time data rate.

Acknowledgement

The authors would like to thank David Garrett for his advice during the initial stages of this work.

References

[1] B. Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Trans. Information Theory*, vol. 51, pp. 389–399, Mar 2003.
[2] G. Knagge, G. Woodward, B. Ninness, and S. Weller, "An optimised parallel tree search for multiuser detection with VLSI implementation strategy," in *IEEE GLOBECOM*, p. to appear, Dec 2004.
[3] G. Knagge, G. Woodward, S. R. Weller, and B. Ninness, "A VLSI optimised parallel tree search for mimo," in *Australian Telecommunications Theory Workshop (AusCTW) 2005*, p. to appear, Feb 2005.
[4] G. Knagge, D. C. Garrett, S. Venkatesan, and C. Nicol, "Matrix datapath architecture for an iterative 4x4 MIMO noise whitening algorithm," in *ACM Great Lakes Symposium on VLSI Circuits (GLSVLSI)*, pp. 590–593, Apr 2003.
[5] L. M. Davis, "Scaled and decoupled cholesky and QR decompositions with application to spherical MIMO detection," *IEEE Wireless Communications & Networking Conference (WCNC)*, pp. 326–331, Mar 2003.
[6] H. Anton and C. Rorres, *Elementary Linear Algebra*. Wiley, 7th ed., 1994.