Low Latency QRD Algorithm
For Future Communication

Lirui Chen, Cang Liu, Yu Wang, Zuocheng Xing, Yang Zhang, Non member, and Jing Liu, student member
National Laboratory for Parallel and Distributed Processing,
National University of Defense Technology, Changsha, China
a) chenlirui14@nudt.edu.cn

Abstract: QR decomposition (QRD) has been a vital component for various baseband processing algorithms, and is one potential bottleneck for next generation (5G) high-performance MIMO systems. To ulteriorly optimize the processing latency (PL) of QRD hardware architecture, this letter proposes a novel anticipated MGS (AMGS) algorithm based on conventional MGS algorithm. Anticipated computing is proposed in AMGS to diminish the PL. Moreover, Reciprocal square root (RSR) algorithm is designed to eliminate the complex operations (dividing and square root), making AMGS algorithm suit more for baseband processors. To evaluate the performance of the proposed AMGS algorithm, the corresponding triangular systolic array (TSA) hardware architecture is also implemented based on AMGS algorithm, whose working frequency is up to 417MHz in 0.13um CMOS technology to decompose a 4×4 real matrix in 31 clock cycles. The implementation results show that the PL performance is superior to other similar works of the literatures we know.

Keywords: QR decomposition, MIMO systems, AMGS.

Classification: Integrated circuits

References
1 Introduction

Multi-input multi-output (MIMO) technology is a promising solution to the rising demand for wireless data service because of its high spectrum efficiency and communication capacity [1]. However, in the coming era of the 5th generation (5G) communication where signal processing latency (PL: time taken from data input to data output) is put to predominant place, the application of MIMO technology is confronted with severe challenges due to its high computational complexity for hardware implementation. Therefore, it’s imperative to design efficient algorithms for hardware architecture in MIMO systems to accommodate the rather rising data rate of future wireless communication systems.

In MIMO systems, QRD is widely utilized in various MIMO detection and channel preprocessing algorithms to reduce the complexity of MIMO detection process and improve the system efficiency. QRD can simplify the procedure of linear MIMO detection algorithms through avoiding the complex inversion calculation of matrices in ZF, MMSE detection algorithms and some other Successive Interference Cancellation (SIC) algorithms. More importantly, in the famous spherical decoder (SD) MIMO detection algorithm [2], whose efficiency is near the theoretically optimal algorithm while the hardware complexity is in acceptable threshold, QRD is an indispensable component to determine the communicating rate and system accuracy. Therefore, there is urgent need for various transceiver processors to improve the hardware performance of QRD component in MIMO systems.

In wireless communication systems, QRD is mainly applied to transfer a channel response matrix $H$ into an upper triangular matrix $R$ and an unitary matrix $Q$. The majority of QRD algorithms are mainly based on
three original algorithms – Householder transformation (HT) algorithm [3], modified Gram-Schmidt (MGS) algorithm [4], and Givens rotation (GR) algorithm [5-9]. As the HT algorithm is too complex to be implemented in hardware, the existing QRD architectures are mainly based on GR algorithm and MGS algorithm in MIMO systems. GR algorithm mainly consists of various trigonometry operations. Because the CORDIC modules in GR algorithms can work in high frequency, the QRD architecture based on GR algorithm can easily achieve high throughput and simultaneously low hardware overhead. However, due to its annihilating matrix elements one by one, the PL performance of hardware architecture based on GR algorithm is relatively longer than that based on MGS algorithm. MGS algorithm process matrix column by column, so the hardware architecture based on MGS achieves better PL performance. But as MGS algorithm mainly consists of multiply, add, divide, and sqrt operations, it leads this QRD architecture to higher hardware overhead. Considering the current circumstance that the IC technology has developed rapidly enough to make area no longer the chief restriction, and the urgent demand for low PL performance in the coming 5G communication, MGS algorithm is deemed to be more profitable for future QRD component in wireless communication.

In this letter, an Anticipated MGS (AMGS) algorithm based on conventional MGS algorithm is proposed to optimize the PL performance of QRD hardware architecture. Anticipated computing is proposed in AMGS to optimize the PL performance. Moreover, the complex operations (dividing and square root) of original MGS algorithm are eliminated by the proposed receptacle square root (RSR) algorithm in MGS, which makes AMGS algorithm suits more for baseband processors and become easy to be designed into an instruction in Application Specific Instruction Set Processor (ASIP) designs. In addition, Newton-Raphson iteration is employed in RSR algorithm to increase the system accuracy. More importantly, as the Newton-Raphson iteration is designed to have the same architecture with other equations in RSR, it entails no more hardware overhead while increasing accuracy. To evaluate the performance of the AMGS algorithm, the corresponding hardware architecture based on AMGS is also designed, and a soft simulation system is built to help to select the optimal word length (WL) structure of hardware. The implemented hardware architecture is synthesized in 0.13um CMOS technology, and implementation results show that the proposed AMGS algorithm can work in 417MHz frequency to decompose a 4×4 real matrix in 31 clock cycles. The PL performance is superior to other similar works of the literatures we know. Although the performance of AMGS is tested with 4×4 real matrix, the dimension of this architecture can be easily modulated, and complex matrixes can be converted into real matrixes in QRD algorithms.

The rest of this letter is organized as follows: in section 2, the original MGS algorithm and the proposed AMGS algorithm are presented. In section 3, the corresponding hardware architecture of AMGS is presented. In section 4, the implemented results are analysed and compared with some existing literatures. Finally, section 5 draws the conclusion.
2 Classical MGS and AMGS algorithms

In this section, the conventional MGS algorithm is sketched firstly as the primary model. Then, The AMGS algorithm is proposed based on this model, including anticipated computing technology, RSR algorithm based on Taylor expansion and Newton-Raphson iteration.

2.1 Classical MGS algorithm

Generally, MGS algorithm decomposes an $n \times n$ matrix $H$ into an $n \times n$ unitary matrix $Q$ and an $n \times n$ upper matrix $R$. To facilitate the description, matrix $H$ is assumed to be $4 \times 4$ real matrix. In this letter, $a_{ij}$ and $q_i$ represent the $ith$ element of column vector $a_j$ and the $ith$ column of matrix $Q$ respectively.

As demonstrated in algorithm 1, the classical MGS algorithm is composed of 4 iterations, where $a_{ij}$ denotes the $ith$ column of matrix $H$ in the $jth$ iteration. In each iteration, the first column $a_{i1}$ is united to generate the vector norm $r_{i1}$ and unitary vector $q_i$ as line 2-3. Then the following columns $a_{ij}$ are updated via subtracting the $q_i$ component to generate $a_{i+1}j$ as line 4-5.

Algorithm 1: Classical MGS Algorithm

| 1: assume $H = \{a_{11}, a_{12}, a_{13}, a_{14}\} $ |
| 2: $r_{11} = |a_{11}| = \sqrt{(a_{11}^2 + (a_{21}^2 + (a_{31}^2 + (a_{41}^2)^2))$ |
| 3: $q_1 = \frac{a_{11}}{r_{11}}$ |
| 4: $r_{12} = (q_1)^T a_{12}, (i = 2, 3, 4)$ |
| 5: $a_{i2} = a_{i1} - r_1 q_1, i = 2, 3, 4$ |
| 6: $r_{22} = |a_{22}| = \sqrt{(a_{21}^2 + (a_{22}^2 + (a_{32}^2 + (a_{42}^2)^2))$ |
| 7: $q_2 = \frac{a_{22}}{r_{22}}$ |
| 8: $r_{23} = (q_2)^T a_{23}, i = 3, 4$ |
| 9: $a_{i3} = a_{i2} - r_2 q_2, i = 3, 4$ |
| 10: $r_{33} = |a_{33}| = \sqrt{(a_{31}^2 + (a_{32}^2 + (a_{33}^2 + (a_{43}^2)^2))$ |
| 11: $q_3 = \frac{a_{33}}{r_{33}}$ |
| 12: $r_{34} = (q_3)^T a_{34}, i = 4$ |
| 13: $a_{i4} = a_{i3} - r_3 q_3, i = 4$ |
| 14: $r_{44} = |a_{44}| = \sqrt{(a_{41}^2 + (a_{42}^2 + (a_{43}^2 + (a_{44}^2)^2))$ |
| 15: $q_4 = \frac{a_{44}}{r_{44}}$ |

2.2 AMGS algorithm

Algorithm 1 shows that, every element relies on the previous element for self calculation, so this algorithm can only run in absolutely serial mode. To increase parallelism, The proposed AMGS algorithm modulated the method of updating $a_{ij}$ to an anticipated computing as (1), where $a_{ij}$ starts self cal-
calculation immediately after getting \( a^1_1 \) and \( a^1_i \), without waiting for \( q^1_1 \).

\[
a^2_i = a^1_i - r^1_1q^1_1 = a^1_i - \left( a^1_i^T a^1_1 \right) \frac{a^1_1}{r^1_1} \tag{1}
\]

In (1), while calculating \( r^1_1 \), the value of \( a^1_1 \cdot a^1_i \) can be calculated simultaneously. Moreover, after getting \( r^1_1, q^1_1 \) and \( r^1_i \) can be generated concurrently. Therefore, the parallelism of this QRD algorithm is promoted.

However, the calculation of \( 1/r^2_{ii} \), \( a^2_j \) and \( q^1_i \) entails square root and dividing operations, which lead to severe PL and may not be supported by some application-specification processors. For further optimization, RSR module is designed to calculate \( 1/r^2_{ii} \) from the value of \( r^2_{ii} \) which can be easily generated from \((a^1_i)^T (a^1_i)\) as (2).

\[
RSR(r^2_{ii}) = \frac{1}{\sqrt{r^2_{ii}}} \tag{2}
\]

With RSR module, the value of \( 1/r^2_{ii} \) is easily generated, thus \( r^2_{ii} \) can be generated from \( 1/r^2_{ii} \) multiplied by \( r^2_{ii} \), therefore both dividing and square root operations are eliminated in AMGS algorithm. So far the proposed AMGS algorithm can be demonstrated as Algorithm 2.

Next we introduce the algorithm of RSR module, which functions as (3)

\[
RSR(z) = \frac{1}{\sqrt{z}} \tag{3}
\]

In RSR module, variable \( z \) is firstly scaled to interval \((0.5, 2]\), marked as \( x \) via shifting by \( 2n \) bits as (4).

\[
x = z \cdot 2^{2n}, x \in (0.5, 2] \tag{4}
\]

Then \( 1/\sqrt{x} \) is approximated via the second-order Taylor expansion at point \( x_0 \) as (5), where \( x_0 \in (0.5, 2] \).

\[
y_n = f(x) = \frac{1}{\sqrt{x}} = f(x_0) + f'(x)(x - x_0) + \frac{f''(x)(x-x_0)^2}{2}
\]

\[
= \frac{3}{8x_0^{1.5}} x^2 - \frac{5}{4x_0^{1.3}} x + \frac{15}{8x_0^{0.5}} \tag{5}
\]

combining (3)(4)(5), RSR can be expressed as (6), where \( a_2, a_1 \) and \( a_0 \) depend on the expansion point \( x_0 \) and \( x_0 \in (0.5, 2] \). Thus the value set of \( a_2, a_1, a_0 \) corresponding to various expansion points of \( x_0 \) in interval \((0.5, 2]\) can be stored in a look-up table (LUT), and the specific value of \( a_2, a_1, a_0 \) in (6) is chosen from LUT according the expansion point of \( x_0 \) nearest to \( x \).

\[
RSR(z) = 2^n \cdot \frac{1}{\sqrt{x}} = 2^n \cdot (a_2x^2 + a_1x + a_0x) \tag{6}
\]

Additionally, to ensure the accuracy of \( 1/\sqrt{x} \), Newton-Raphson iteration is adopted in RSR module. The basic function of Newton-Raphson iteration algorithm \( \varphi(y) \) is defined as (7), thus \( 1/\sqrt{x} \) is an accurate root of \( \varphi(y) \). Let \( y_n \)
denote the approximate value of $1/\sqrt{x}$ generated from (5), then $y_n$ is closed to the accurate root of (7). According to the Newton-Raphson iteration equation (8), $y_{n+1}$ is more closed to this root, namely more closed to the accurate value of $1/\sqrt{x}$.

$$\varphi(y) = \frac{1}{y^2} - x$$

(7)

$$y_{n+1} = y_n - \frac{\varphi(y_n)}{\varphi'(y_n)} = \frac{3}{2} y_n - \frac{x}{2 y_n^3}$$

(8)

As the Newton-Raphson iteration equation (8) is similar with (5) in scheme, the hardware for (5) can be reused to calculate (8). Therefore the Newton-Raphson iteration entails no more hardware overhead while increase accuracy.

To make clear the superiority of the AMGS algorithm, the critical paths of MGS and AMGS are demonstrated and compared in Fig.1 and Fig.2. It’s clear that the Parallelism of AMGS is significantly improved than MGS, so the AMGS algorithm has the potential to improve the PL performance. More importantly, the complex operations (dividing and square root) are eliminated in AMGS algorithm, which makes this algorithm more applicable to most platforms and can be easily packed as an instruction in application specific instruction set processor (ASIP).
Algorithm 2: AMGS Algorithm

1: assume : \( H = \{a_i^1, a_i^2, a_i^3, a_i^4\} \)
2: \( r_{ii}^2 = (a_i^1)^T a_i^1 \), \( p_{ii} = (a_i^1)^T a_i^1 \) (i = 2, 3, 4)
3: \( \frac{1}{r_{ii}} = RSR(r_{ii}^2) \)
4: \( q_i = a_i^1(\frac{1}{r_{ii}}) \), \( r_{ii} = \frac{p_{ii}}{r_{ii}^2} \)
5: \( r_{ii} = r_{ii}^2(\frac{1}{r_{ii}}) \), \( a_i^2 = a_i^1 - r_{ii}q_i \)
6: \( r_{ii}^2 = (a_i^2)^T a_i^2 \), \( p_{2i} = (a_i^2)^T a_i^2 \) (i = 3, 4)
7: \( \frac{1}{r_{2i}} = RSR(r_{2i}^2) \)
8: \( q_2 = a_i^2(\frac{1}{r_{2i}}) \), \( r_{2i} = \frac{p_{2i}}{r_{2i}^2} \)
9: \( r_{2i} = r_{2i}^2(\frac{1}{r_{2i}}) \), \( a_i^3 = a_i^2 - r_{2i}q_2 \)
10: \( r_{3i}^2 = (a_i^3)^T a_i^3 \), \( p_{3i} = (a_i^3)^T a_i^3 \) (i = 4)
11: \( \frac{1}{r_{3i}} = RSR(r_{3i}^2) \)
12: \( q_3 = a_i^3(\frac{1}{r_{3i}}) \), \( r_{3i} = \frac{p_{3i}}{r_{3i}^2} \)
13: \( r_{3i} = r_{3i}^2(\frac{1}{r_{3i}}) \), \( a_i^4 = a_i^3 - r_{3i}q_3 \)
14: \( r_{4i}^2 = (a_i^4)^T a_i^4 \)
15: \( \frac{1}{r_{4i}} = RSR(r_{4i}^2) \)
16: \( q_4 = a_i^4(\frac{1}{r_{4i}}) \), \( r_{4i} = r_{4i}^2(\frac{1}{r_{4i}}) \)

3 Hardware architectures

To evaluate the performance the proposed AMGS algorithm, we designed the corresponding hardware based on AMGS. This section demonstrates the STA hardware architecture based on AMGS algorithm to decompose 4×4 real matrices. According to the scheme of AMGS algorithm, the corresponding hardware architecture is composed of two basic blocks: Block1 and Block2. Block1 is used to calculate \( 1/r_{ii}, q_i, \) and \( r_{ii} \). And Block2 is used to calculate \( a_i^q \) for the next iteration.

3.1 TSA hardware architecture of AMGS

As shown in Fig.3. The overall TSA architecture based on AMGS algorithm is composed of two basic blocks: Block1 and Block2, which are elaborated in the following parts of this section. On the timing sequence, the whole procedure is divided into 4 iterations to decompose a 4×4 real matrix. And in each iteration, Block1 and Block2 work in parallel to accomplish the tasks defined as (9) and (10), respectively. The implemented TSA architecture takes 31 clock cycles to complete one QRD calculation, and can work in pipeline to generate results every three clock cycles.

The major advantage of this TSA architecture is its parallelism and pipeline property. With this architecture, decomposing a \( n \times n \) real matrix only needs \( n \) time iterations, while in the case of classical MGS the time needed is up to \( (2n - 1) \) time iterations. So the architecture based on AMGS has the potential to reduce PL than that based on MGS.

\[ r_{ii} = |a_i|, \quad q_i = \frac{a_i}{r_{ii}}, \quad RSR(r_{ii}^2) = \frac{1}{r_{ii}} \]  (9)

\[ r_{ij} = (a_i^m \cdot a_j^m) RSR(r_{ii}^2), \quad a_j^{m+1} = a_j^m - r_{ij}q_i \]  (10)
3.2 Basic blocks

The hardware architecture of Block1 is presented as Fig.4, which contains 8 multipliers, an adder and an RSR module to accomplish the tasks defined in (9). Block1 is used to normalize the vector $a_i$ to generate the unitary vector $q_i$ and parameter $1/r_{ii}$ for Block2’s calculation. In Fig.4, $q_{ij}$ ($j=1,2,3,4$) denotes the $j$th element of vector $q_i$, and the shadowed components indicate the shared hardware. Because the calculation of $r_{ii}$ is outside of the critical paths, it shares one multiplier with other modules to save hardware overhead. Finally, Block1 takes 8 clock cycles to generate the vector of $q_i$ and one more clock cycle to generate a non-critical result $r_{ii}$.

The RSR module is a crucial component in Block1 to realize the function of (3), as demonstrated in Fig.5. In RSR, the input signal $z$ is connected with $r_{ii}^2$ from the adder output register, and the output signal $y_{n+1}$ is sampled as $RSR(r_{ii}^2)$ by the followed multipliers. The scale module functions as (3), and the PW module functions as a look-up table (LUT) to store a series of Taylor expansion series $a_2, a_1, a_0$ in different expansion points. When $x$ is input into PW, the expansion series $a_2, a_1, a_0$ of the expansion point which is nearest to $x$ are generated for the following calculations. The RSR takes 4 clocks to pass one data through the whole processing, including the rough calculation via Taylor expansion (3)(5), and the accurate calculation by Newton-Raphson iteration (7). In the first clock, signal $x$ is squared via multiplier-2 and $a_2, a_1, a_0$ are generated from PW module, then in the second clock the squared $x$ is multiplied by $a_2$ and $x$ is multiplied with $a_1$, thus $y_n$ is generated through an adder. In the third clock, $y_n$ is sent back and squared by multiplier-2. Finally, in the fourth clock, the squared $y_n$ is multiplied by $x$ in multiplier-3 and $y_n$ is multiplied with $3/2$ via the shifter and adder. The accurate result $y_{n+1}$ is generated from a subtractor in the fourth clock. In RSR module, as the Newton-Raphson iteration equation (8) is designed to have the same scheme with (5), the calculation of $y_{n+1}$ can reuse the same hardware of calculating $y_n$. Therefore, the Newton-Raphson iteration entails no more hardware while increasing accuracy.

The hardware architecture of Block2 is demonstrated in Fig.6, which contains 7 multipliers, and 5 adders to accomplish the task defined by (10).
Block2 is used to eliminate the $a^m_i$ component from vector $a^m_j$ to generate the vector $a^{m+1}_j$ for next iteration. In Block2, as the data $1/r_{ii}$ comes relatively later from Block1 due to the long PL of RSR module, there is no immediate demand for the result of $(a^m_i \cdot a^m_j)$. So the 4 multiplying operations share two multipliers to be accomplished in two clocks in order to save hardware. Eventually, the total PL of Block2 is 8 clock cycles.

Fig. 4. hardware architecture of Block1

Fig. 5. hardware architecture of RSR

Fig. 6. hardware architecture of Block2

4 Implementation results and comparison

This section first introduces our method of constructing the optimal word length (WL) format in the hardware. Then the ultimate performance of the hardware implementation is presented and compared with other existing works. Additionally, numerous rules are defined to fairly quantify the performance of various works in this section.

In order to evaluate the hardware quality with various fix-point WL format, a simulation system based on AMGS is designed for the hardware,
in which ten thousand sample results \((Q_i, R_i)\) are generated from decomposing the random \(4 \times 4\) matrixes in each simulation. Simultaneously, the reference results are generated from decomposing the same matrixes in double-floating accuracy. Then each sample result is subtracted by corresponding reference result to generate the absolute error matrixes \(\varepsilon_{Q_i}\) and \(\varepsilon_{R_i}\). Finally the average errors of \(Q_i\) and \(R_i\) are generated by averaging all the absolute data in every \(\varepsilon_{Q_i}\) and \(\varepsilon_{R_i}\), signed as \(\varepsilon_Q\) and \(\varepsilon_R\) respectively.

The optimal numbers of integer bits and fractional bits of the WL format are determined by the simulation results as illustrated in Fig.7 and Fig.8. In Fig.7, the length of integer is set to 6, which is much longer than the selected length, while the number of fractional bits changes from 5 to 16 to observe the error variation of matrix \(R\). As shown in Fig.7, the precision hardly increase anymore after the fractional bits reaches 13, thus the number of fractional bits is determined. In Fig.8, the number of fractional bits is set to 13, while the total WL changes from 14 to 20. According the this figure, the appropriate number of WL is 16. Therefore, the optimal WL format of the hardware is determined to 16 bits, including 13 fractional bits, 2 integer bits and one sign bit.

To evaluate the ultimate performance of the designed QRD hardware architecture, the TSA architecture based on AMGS algorithm is implemented.
Table I. performance comparison of QRD

<table>
<thead>
<tr>
<th></th>
<th>Alg</th>
<th>Tech</th>
<th>Matrix Size</th>
<th>CL</th>
<th>GC /k</th>
<th>CP /ns</th>
<th>NPL /ns</th>
<th>NR QRD/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>this</td>
<td>AMGS</td>
<td>0.13um</td>
<td>R4×4</td>
<td>32</td>
<td>238.3</td>
<td>2.4</td>
<td>76.8</td>
<td>138.9M</td>
</tr>
<tr>
<td>[4].a</td>
<td>MGS</td>
<td>0.18um</td>
<td>R4×4</td>
<td>35</td>
<td>32.6</td>
<td>2.5</td>
<td>63.2</td>
<td>5.5M</td>
</tr>
<tr>
<td>[4].b</td>
<td>MGS</td>
<td>0.18um</td>
<td>R4×4</td>
<td>35</td>
<td>55.3</td>
<td>2.5</td>
<td>63.2</td>
<td>15.8M</td>
</tr>
<tr>
<td>[10]</td>
<td>Hybrid</td>
<td>0.13um</td>
<td>C4×4</td>
<td>150</td>
<td>36</td>
<td>3.6</td>
<td>135</td>
<td>27.8M</td>
</tr>
<tr>
<td>[5]</td>
<td>GR</td>
<td>0.18um</td>
<td>C4×4</td>
<td>96</td>
<td>192.1</td>
<td>5</td>
<td>86.7</td>
<td>138.5M</td>
</tr>
<tr>
<td>[6]</td>
<td>Hybrid</td>
<td>65nm</td>
<td>C4×4</td>
<td>87</td>
<td>362</td>
<td>2</td>
<td>87</td>
<td>138M</td>
</tr>
<tr>
<td>[7]</td>
<td>CGRA</td>
<td>65nm</td>
<td>C4×4</td>
<td>C</td>
<td>1055</td>
<td>2</td>
<td>C</td>
<td>79.2M</td>
</tr>
<tr>
<td>[8]</td>
<td>GR</td>
<td>0.18um</td>
<td>R8×8</td>
<td>C</td>
<td>152</td>
<td>10</td>
<td>C</td>
<td>138.5M</td>
</tr>
<tr>
<td>[9]</td>
<td>GR</td>
<td>90nm</td>
<td>C4×4</td>
<td>C</td>
<td>437.5</td>
<td>8.6</td>
<td>C</td>
<td>80.3M</td>
</tr>
</tbody>
</table>

with SMIC 0.13um CMOS technology to decompose the 4×4 real matrix. Results show that the latency of critical path is 2.4ns. The clock latency (CL), which denotes the clock counts for a data to travel from input to output, is 31 clock cycles. And a total of 328k primitive gates is cost in the design. To fairly compare the latency performance with other existing works, the normalized processing latency (NPL) is defined as (11).

\[
NPL = \left( \frac{PL}{\eta} \right) \cdot \frac{4 \times 4}{M \times N} \cdot \frac{1}{\eta} \cdot \frac{0.13um}{tech} \tag{11}
\]

Where \( M \times N \) is the size of decomposed matrix. When the matrix is a complex matrix, the workload equals to decomposing a real matrix of the double size, thus the parameter \( \eta \) is 4, otherwise \( \eta \) is 1. Additionally, the technology parameter \( tech \) is normalized to 0.13um CMOS technology. In the similar way, the normalized QRD rate is also defined to evaluate the throughput performance as (12).

\[
NR = Rate \cdot \frac{M \times N}{4 \times 4} \cdot \frac{tech}{0.13um} \cdot \eta \tag{12}
\]

Where the Rate denotes the number of matrixes decomposed per second.

Table I illustrates the comparison results of the proposed TSA architecture with numerous existing QRD works. In table I, AMGS and [4] are both based on MGS algorithm. Chang et al. [4] implemented a TSA architecture and iteration architecture to decompose 4 × 4 real matrixes, which cost merely 55.3k GC and 32.6k GC respectively and reached an NPL of 63.2ns. However, its NR performance is not so good. Compared with [4], this design inherited the excellent NPL feature of MGS and achieves higher NR performance. Although the NPL of AMGS is slightly higher than [4], we should consider the fact that the WL of [4] is 14bit while the WL of AMGS is 16bit. Experiments show that when using 14bit WL, the proposed AMGS can achieve a better NPL performance than [4]. More importantly, the well-defined WL format based on the hardware simulating system also makes our design more accurate and reliable than that of [4].
The QRD architectures in [5,7-9] are based on GR algorithm, and [6,10] are based on Hybrid algorithm. Lin JS [5] proposed a two-phase QRD architecture, which incorporated a single exchange of the column and arrow into the general QRD procedure to achieve a better NR performance of 138.5M. Zhang C [6] designed a hardware architecture based on hybrid GR algorithm to increase the parallelism, which achieves a good NR performance of 138M and relatively low NPL of 87ns. As demonstrated in table I, one common architecture of GR is its high NR performance deriving from its high working frequency. In our design, as long-latency operations (dividing, square root) are eliminated, the hardware based on AMGS can work in higher frequency conditions to reach better NR performance.

Comparing the two indicators (NPL and NR), we believe the NPL performance is more crucial for a design. Because most hardware design with low NR performance could enhance its throughput by pipeline technique or adding more pipeline registers in the same semiconductor technology condition. More importantly, in the future 5G communication, the QR results is immediately waited after getting channel matrixes for the coding process, which also makes the NPL more crucial than NR performance. The proposed hardware architecture based on AMGS achieves an NPL performance of 76.8ns in 0.13\(\mu\)m CMOS technology with 16bit WL structure, superior to other similar works of literatures we know.

### 5 Conclusion

To accommodate the urgent demand for low latency in the coming 5G wireless communication, this letter proposed an AMGS algorithm for QRD, one vital component of MIMO systems. Anticipated computing is proposed in AMGS algorithm to optimize the PL performance. Moreover, the complex operations (dividing and square root) in original MGS algorithm are eliminated by RSR algorithm in AMGS, which makes AMGS easy to be designed into an instruction in ASIP design for future communication processors. To evaluate the performance of the proposed AMGS algorithm, the corresponding hardware based on AMGS are also designed, and the parameter NPL and NR are defined in this letter to compare the performance of AMGS with other similar works. The synthesis results show that the proposed AMGS algorithm achieves an NPL performance of 76.8ns and an NR performance of 138.9M when implemented with 16bit WL in 0.13\(\mu\)m CMOS technology, superior to other similar works of literatures we know. Although the performance of AMGS is evaluated with the hardware architecture of 4\(\times\)4 real matrix, the dimension of this architecture can be easily modulated to other dimensional matrixes in MIMO systems, such as 8\(\times\)8, 16\(\times\)16, and the complex matrix can be converted to real matrixes in QRD algorithms.