Architectural Optimization of Decomposition Algorithms for Wireless Communication Systems

Ali Irturk[†], Bridget Benson[†], Nikolay Laptev[‡], Ryan Kastner[†]

***Abstract*— Matrix decomposition is required in various algorithms used in wireless communication applications. FPGAs strike a balance between ASICs and DSPs, as they have the programmability of software with performance capacity approaching that of a custom hardware implementation. However, FPGA architectures require designers to make a countless number of system, architectural and logic design decisions. By performing design space exploration, a designer can find the optimal device for a specific application, however very few tools exist which can accomplish this task. This paper presents automatic generation and optimization of decomposition methods using a core generator tool, GUSTO, that we developed to enable easy design space exploration with different parameterization options such as resource allocation, bit widths of the data, number of functional units and organization of controllers and interconnects. We present a detailed study of area and throughput tradeoffs of matrix decomposition architectures using different parameterizations.**

I. Introduction

Matrix decompositions are essential computations for simplifying and reducing the computational complexity of various algorithms used in wireless communication. For example, decomposition methods are used for simplifying matrix inversion which are used in MIMO-OFDM systems' minimum mean square error algorithms for pre-coding in spatial multiplexing [1], equalization algorithms to remove the effect of the channel on the signal [2] and detection-estimation algorithms in space-time coding [3].

The choice of computational platform plays a significant role in the overall design and implementation of wireless communication systems. A designer should determine an appropriate platform between a wide range of hardware: Application Specific Integrated Circuits (ASICs) and software: Digital Signal Processors (DSPs). ASICs offer exceptional performance results at the price of long time to market and high non-recurring engineering (NRE) costs. On the other hand, DSPs ease the development of these architectures and offer a short time to market, however they lack the performance capacity for high throughput applications. Field Programmable Gate Arrays (FPGAs) strike a balance between ASICs and DSPs, as they have the programmability of software with performance capacity approaching that of a custom hardware implementation. FPGAs present designers with substantially more parallelism allowing more efficient application implementation. FPGAs are becoming an increasingly common platform for wireless communication [4-6] as they provide powerful computational architectural features such as vast amounts of programmable logic elements, embedded multipliers, shift register LUTs (SRLs), Block RAMs (BRAMs), DSP blocks and Digital Clock Managers (DCMs).

When building an FPGA architecture, designers need to make a countless number of system, architectural and logic design decisions with regards to resource allocation, bit widths of the data, number of functional units and organization of controllers and interconnects. The main goal is to deliver the smallest, fastest device for the application that uses the least power. However achieving all of these goals in one design is at times contradictory since designing a faster device also frequently results in a larger, more power hungry device. By performing design space exploration, a designer can find the optimal device for a specific application, however very few tools exist which can accomplish this task. To ease the design space exploration of different matrix computations, we design an easy to use tool, GUSTO (**"G**eneral architecture design **U**tility and **S**ynthesis **T**ool for **O**ptimization")[7, 8]. GUSTO takes 5 inputs, namely the operation type (QR, LU, Cholesky decompositions), the matrix dimension, the type and number of arithmetic resources, data representation (the integer and fractional bit width) and modes of operation. GUSTO has two different modes of operation: Mode 1 and Mode 2. Mode 1 automatically generates a general purpose architecture and Mode 2 optimizes the general purpose architecture creating an application specific architecture to improve its area and performance results.

Our major contributions are as follows:

1) *Automatic generation and optimization of different decomposition architectures with parameterized inputs: resource allocation, bit widths, matrix dimensions, modes and methods.*
2) *Comparison of different matrix decomposition methods in terms of different matrix dimensions, bit widths and parallelism.*
3) *Thorough study of area and throughput tradeoffs of matrix decomposition architectures using different parameterizations and a case study: Implementation of Adaptive Weight Calculation Core using QRD-RLS algorithm.*

```
QRD-MGS(A)
1  for i = 1 : n
2      $X_i = A_i$
3  for i = 1 : n
4      $R_{ii} = ||X_i||$
5      $Q_i = X_i / R_{ii}$
6      for j = i+1 : n
7          $R_{ij} = <Q_i, X_j>$
8          $X_j = X_j - R_{ij}Q_i$
```

(a) QRD MGS Algorithm

$$Q=\begin{bmatrix} Q_{11} & Q_{12} & Q_{13} & Q_{14} \\ Q_{21} & Q_{22} & Q_{23} & Q_{24} \\ Q_{31} & Q_{32} & Q_{33} & Q_{34} \\ Q_{41} & Q_{42} & Q_{43} & Q_{44} \end{bmatrix} \quad R=\begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ 0 & R_{22} & R_{23} & R_{24} \\ 0 & 0 & R_{33} & R_{34} \\ 0 & 0 & 0 & R_{44} \end{bmatrix}$$

(b) Resulting Matrices

Fig. 1. QR decomposition (QR-MGS) algorithm is presented in (a). The resulting matrices of the decomposition are shown in (b).

```
LU(A)
1 for j = 1 : n
2   for k = 1 : j-1
3     for i = k+1:j-1
4       A(i,j) = A(i,j) - A(i,k)*A(k,j);
5   for k = 1 : j-1
6     for i = j : n
7       A(i,j) = A(i,j) - A(i,k)*A(k,j);
8   for k = j+1 : n
9     A(k, j) = A(k, j) / A(j,j);
```

(a) LU Algorithm

$$U=\begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{23} & U_{24} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{44} \end{bmatrix} \quad L=\begin{bmatrix} L_{11} & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 \\ L_{31} & L_{32} & L_{33} & 0 \\ L_{41} & L_{42} & L_{43} & L_{44} \end{bmatrix}$$

(b) Resulting Matrices

Fig. 2. LU decomposition algorithm is presented in (a). The resulting matrices of the decomposition are shown in (b).

```
Cholesky(A)
1 for k = 1 : n
2     $G_{kk} = sqrt(A_{kk})$
3     for i = k+1 : n
4         $G_{ik} = A_{ik} / A_{kk}$
5     for j = k+1 : n
6         for t = j : n
7             $A_{tj} = A_{tj} - G_{tk}G_{jk}$
```

(a) Cholesky Algorithm

$$L=\begin{bmatrix} L_{11} & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 \\ L_{31} & L_{32} & L_{33} & 0 \\ L_{41} & L_{42} & L_{43} & L_{44} \end{bmatrix} \quad L^T=\begin{bmatrix} L_{11} & L_{21} & L_{31} & L_{41} \\ 0 & L_{22} & L_{32} & L_{42} \\ 0 & 0 & L_{33} & L_{43} \\ 0 & 0 & 0 & L_{44} \end{bmatrix}$$

(b) Resulting Matrices

Fig. 3. Cholesky decomposition algorithm is presented in (a). The resulting matrices of the decomposition are shown in (b).

This paper is organized as follows. Section II introduces decomposition methods: QR, LU and Cholesky, which are frequently used in wireless communication systems. Section III describes our tool to generate matrix decomposition architectures and the optimizations performed, namely: static architecture generation and resource trimming for optimization. Section IV presents our implementation results in terms of area, timing and throughput as well as a case study that implements Adaptive Weight Calculation and compares our results with previously published work. We conclude in Section V.

II. Matrix Decomposition Methods

Decomposition methods such as QR, LU and Cholesky, provide analytic simplicity and computational convenience for many different wireless communication algorithms. The selection of the decomposition method depends on the characteristics of the given matrix. As an example in matrix inversion, QR decomposition is used to generate an equivalent upper triangular system for non-square matrices. We further explain these decomposition methods, their characteristics and algorithms, and the resulting matrices in the next subsections [9]. Note that for square matrices, n denotes the matrix size (i.e. matrix with $n = 4$ implies a 4×4 matrix). For rectangular matrices, m and n denote the number of rows and columns in the matrix respectively (i.e. matrix with $m = 3, n = 4$ denotes a 3×4 matrix).

A. QR Decomposition

QR decomposition is an elementary operation, which decomposes a matrix into an orthogonal and a triangular matrix. Given $A \in \mathbb{R}^{m\times n}$ with $rank(A) = n$, QR decomposition exists as $A = Q\times R$ where $Q \in \mathbb{R}^{m\times n}$ has orthonormal columns, $Q^T \times Q = Q \times Q^T = I$, $Q^{-1} = Q^T$, and $R \in \mathbb{R}^{n\times n}$ is an upper triangular matrix (Figure 1(b)).

There are three different QR decomposition methods: Gram-Schmidt orthogonormalization (Classical or Modified), Givens Rotations (GR) and Householder Reflections. QRD-Modified Gram-Schmidt (MGS) is a slightly modified version of the QRD-Classical Gram-Schmidt (CGS) algorithm. QRD-MGS is numerically more accurate and stable than QRD-CGS and numerically equivalent to the Givens Rotations solution [10-12] (the solution that has been the focus of previously published hardware implementations because of its stability and accuracy). If the input matrix, A, is well-conditioned and non-singular, the resulting matrices, Q and R, satisfy their required matrix characteristics and QRD-MGS results are accurate to floating-point machine precision [12]. We therefore present the QRD-MGS algorithm in Figure 1(a) and describe it below.

A, Q, R and X are the input, orthogonal, upper triangular and intermediate matrices, respectively. The intermediate matrix is the updated input matrix throughout the solution steps. Matrices with only one index as A_i or X_j represent the columns of the matrix and matrices with two indices like R_{ij} represent the entry at the intersection of *ith* row with *jth* column of the matrix where $1 \leq i,j \leq n$.

In Figure 1(a) we show that we start every decomposition by transferring the input matrix columns, A_i, into the memory elements (2). Diagonal entries of the R matrix are the Euclidean norm of the intermediate matrix columns which is shown as (4). The Q matrix columns are calculated by the division of the intermediate matrix columns by the Euclidean norm of the intermediate matrix column, which is the diagonal element of R (5). Non-diagonal entries of the R matrix are computed by projecting the Q matrix columns onto the intermediate matrix columns one by one (7) such that after the solution of Q_2, it is projected onto X_3 and X_4 to compute R_{23} and R_{24}. Lastly, the intermediate matrix columns are updated by (8).

B. LU Decomposition

If A is a square matrix, $A \in \mathbb{R}^{n\times n}$, and its leading principal submatrices are all nonsingular, $det(A(1{:}k, 1{:}k)) \neq 0$ for $k = 1 : n\text{-}1$, matrix A can be decomposed into unique lower triangular and upper triangular matrices. LU decomposition of a matrix A is shown as $A = L \times U$, where L and U are the lower and upper triangular matrices respectively (Figure 2(b)).

The LU algorithm is shown in Figure 2(a). It writes lower and upper triangular matrices onto the A matrix entries. Then it updates the values of the A matrix column by column ((4) and (7)). The final values are computed by the division of each column entry by the diagonal entry of that column (9).

C. Cholesky Decomposition

Cholesky decomposition is another elementary operation, which decomposes a symmetric positive definite matrix, $A \in \mathbb{R}^{n\times n}$, into a unique lower triangular matrix, $G \in \mathbb{R}^{n\times n}$, with positive diagonal entries. A matrix $A \in \mathbb{R}^{n\times n}$ is *positive definite* if $x^TAx > 0$ for $x \in \mathbb{R}^n$ and $x \neq 0$ where if A is *symmetric positive definite* then $A^T = A$. A positive definite matrix is always nonsingular and its determinant is always positive. Cholesky decomposition of a matrix A is shown as $A = G\times G^T$, where G is a unique lower triangular matrix, Cholesky triangle, and G^T is the transpose of this lower triangular matrix (Figure 3(b)).

Figure 3(a) shows the Cholesky decomposition algorithm. We start decomposition by transferring the input matrix, A, into the memory elements. The diagonal entries of lower triangular matrix, G, are the square root of the diagonal entries of the given matrix (2). We calculate the entries below the diagonal entries by dividing the corresponding element of the given matrix by the belonging column diagonal element (4). The algorithm works column by column and after the computation of the first column of the diagonal matrix with the given matrix entries, the elements in the next columns are updated (7). For example after the computation of G_{11} by (2), G_{21}, G_{31}, G_{41} by (4), the second column: A_{22}, A_{32}, A_{42}, third column: A_{33}, A_{43}, and fourth column: A_{44} are updated by (7).

III. GUSTO : General Architecture Design Utility and Synthesis Tool for Optimization

When implementing decomposition methods in hardware, there are different architectural and design decisions that a designer needs to make such as choosing the decomposition method, matrix dimension, number of bits, number of arithmetic resources, and organization of controllers and interconnects. Matrix dimension changes with the number of antennas that are employed, the number of bits defines the precision of the hardware, and the number of arithmetic resources affects the area as well as the throughput of the design. Design space exploration allows us to set various options, such as number of bits and arithmetic resources used, in order to find the optimal design for a given application and therefore it is a critical step in design process. With an increasingly strict time to market, designers need to perform design space exploration and do the necessary optimizations to the original architecture in a short amount of time. The design of a tool which provides automatic generation and optimization of these architectures is therefore crucial to the success of an efficient design.

Therefore, we design a tool, GUSTO "**G**eneral architecture design **U**tility and **S**ynthesis **T**ool for **O**ptimization," which automatically generates and optimizes hardware for matrix computations. GUSTO offers different inputs to the user such as decomposition method, matrix dimension, number of arithmetic resources, number of integer and fractional bit width, and two modes of operation (Mode 1 or Mode 2) as shown in Figure 4. This automatic generation and optimization provides fast results to the designer, so the designer can consider different architectural implementations and choose the most suitable one for his needs. GUSTO creates architectures which use fixed-point arithmetic and provides an error analysis after the resource allocation step to find an appropriate fixed point representation with similar accuracy to

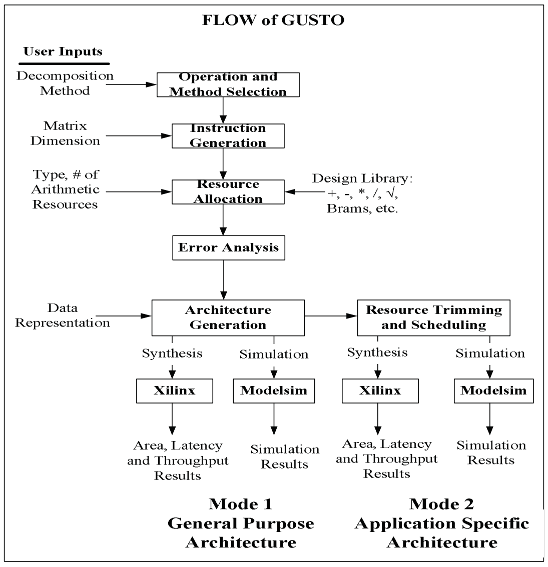
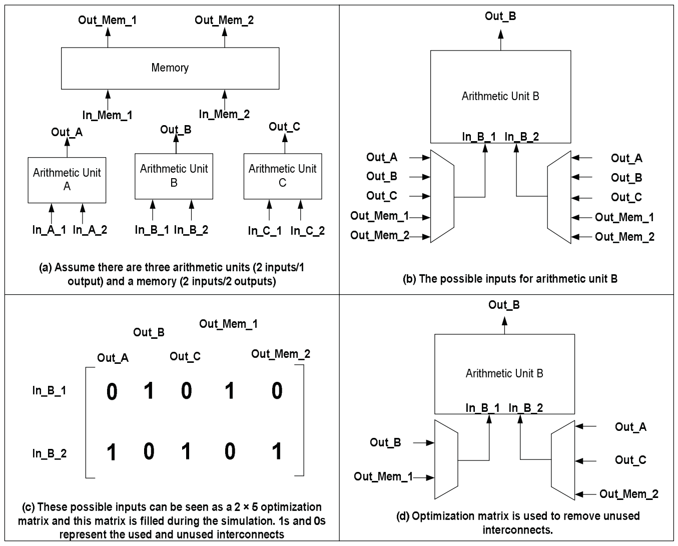

FLOW of GUSTO
User Inputs
Decomposition Method
Operation and Method Selection
Matrix Dimension
Instruction Generation
Type, # of Arithmetic Resources
Resource Allocation
Design Library: +, -, *, /, √, Brams, etc.
Error Analysis
Data Representation
Architecture Generation
Resource Trimming and Scheduling
Synthesis
Simulation
Xilinx
Modelsim
Area, Latency and Throughput Results
Simulation Results
Mode 1 General Purpose Architecture
Mode 2 Application Specific Architecture

Fig. 4. Flow and modes of GUSTO.

that of a floating point implementation. The error analysis step uses the user defined input data, and performs decomposition with variable bit-length fixed-point and floating-point (single or double precision) arithmetic, and presents error in terms of mean error, standard deviation of error and mean percentage of error. Error is defined as the difference between fixed-point and floating-point arithmetic results.

Mode 1 of GUSTO creates a general purpose architecture using resource constrained list scheduling. The benefit of a general purpose architecture is that it is capable of performing all three decomposition methods with a selection input provided to the user. A general purpose architecture is useful if the determination of which decomposition method to use is not possible beforehand. However general purpose architectures come at a price of larger area, lower throughput and higher power consumption. Since general purpose architectures do not lead to high performance results, it is best to create an application specific architecture for a specific decomposition method by optimizing the general purpose architecture.

Mode 2 of GUSTO creates an application specific architecture for a specific decomposition method. The architecture becomes scheduled, static and optimized while maintaining the correctness of solution. Application specific architecture implementation is performed in two steps: *generating a scheduled and static architecture*, and *trimming the unused resources from the architecture for further optimization*. After generation of the general purpose architecture, this architecture is simulated by Mode 2 of GUSTO to gather information for the assignments done to the memory elements and arithmetic units during the scheduling process. This information is used in the application specific architecture to cancel the scheduling process and dynamic memory assignments making the architecture scheduled and static which provides significant area and timing savings.



Out_Mem_1
Out_Mem_2
Memory
In_Mem_1
In_Mem_2
Out_A
Out_B
Out_C
Arithmetic Unit A
Arithmetic Unit B
Arithmetic Unit C
In_A_1 In_A_2
In_B_1 In_B_2
In_C_1 In_C_2
(a) Assume there are three arithmetic units (2 inputs/1 output) and a memory (2 inputs/2 outputs)
Out_B
Arithmetic Unit B
In_B_1 In_B_2
Out_A
Out_B
Out_C
Out_Mem_1
Out_Mem_2
(b) The possible inputs for arithmetic unit B
Out_A Out_B Out_C Out_Mem_1 Out_Mem_2
In_B_1 0 1 0 1 0
In_B_2 1 0 1 0 1
(c) These possible inputs can be seen as a 2 × 5 optimization matrix and this matrix is filled during the simulation. 1s and 0s represent the used and unused interconnects
Out_B
Arithmetic Unit B
In_B_1 In_B_2
Out_B
Out_Mem_1
Out_A
Out_C
Out_Mem_2
(d) Optimization matrix is used to remove unused interconnects.

Fig. 5. Trimming feature of GUSTO.

GUSTO also performs optimization by trimming the unused resources and interconnects to improve its area results as well as throughput. GUSTO defines the connections between the resources with optimization matrices, simulates the architecture to fill up the optimization matrices with 1s and 0s denoting used and unused resources respectively and trims the unused resources away. The trimming process is shown with an example in Figure 5 (a,b,c and d). Assume that there are three arithmetic units (2 inputs/1 output) and a memory unit (2 inputs/2 outputs) as shown in (a). The possible inputs to arithmetic unit B are shown in (b). An optimization matrix for arithmetic unit B is created by GUSTO and this matrix is filled up with 1s and 0s through simulation (c). Even though *Out_A*, *Out_B*, *Out_C*, *Out_Mem_1* and *Out_Mem_2* are all inputs to the arithmetic unit B, not all may be used during the decomposition process. As can be seen from the matrix in (c), *Out_A*, *Out_C* and *Out_Mem_2* are never being used as inputs to the *In_B_1* and *Out_B*, *Out_Mem_1* are never being used as inputs to *In_B_2*. GUSTO performs optimization by trimming away these unused interconnects (d). Also it is important to note that if an optimization matrix of an arithmetic unit is filled by 0s, the arithmetic unit can be removed with its interconnects.

IV. Results

This section presents the effectiveness of our tool, GUSTO, by showing its different design space exploration examples. We divided this section into two parts: inflection point analysis and architectural design alternatives analysis. Inflection point analysis presents timing results of decomposition methods in terms of clock cycles for different executions (sequential and parallel), matrix sizes and bit-widths. Inflection point analysis answers at what matrix size does an inflection point occur and how does varying bit width and degree of parallelism change the inflection point for a specific decomposition method? Architectural design alternatives analysis presents area and performance results of different decomposition architectures using different parameterizations. Area and performance results are presented in terms of slices and throughput respectively. This analysis also shows how GUSTO finds the optimal hardware by showing Mode 1 and Mode 2 results of different decomposition methods. Throughput is calculated by dividing the maximum clock frequency (MHz) by the number of clock cycles to perform matrix decomposition. Our designs are written in Verilog and synthesized using Xilinx ISE 9.2. Resource utilization and design frequency are post place and route values obtained using a Virtex 4 SX35 FPGA.

Inflection Point Analysis: We present three different analyses for comparison of decomposition methods in Figure 6 (a), (b) and (c). The total number of operations for different decomposition methods is shown in Figure 6 (a) in log domain. We compare sequential and parallel execution of different decomposition methods for different bit widths (16, 32 and 64 bits) and matrix sizes in Figure 6 (b) and (c) respectively. QR, LU and Cholesky decomposition methods are shown with square, spade and triangle respectively. 16, 32 and 64 bits of bit widths are shown with smaller dashed, dashed and solid lines respectively. We show inflection points between these decomposition methods by balloons.

The total number of operations (a) shows that QR decomposition requires significantly higher number of operations compared to LU and Cholesky decompositions. LU and Cholesky decompositions require a close number of operations where LU decomposition requires more operations than Cholesky decomposition after 4 × 4 matrices. The sequential execution of decomposition methods (b) show that

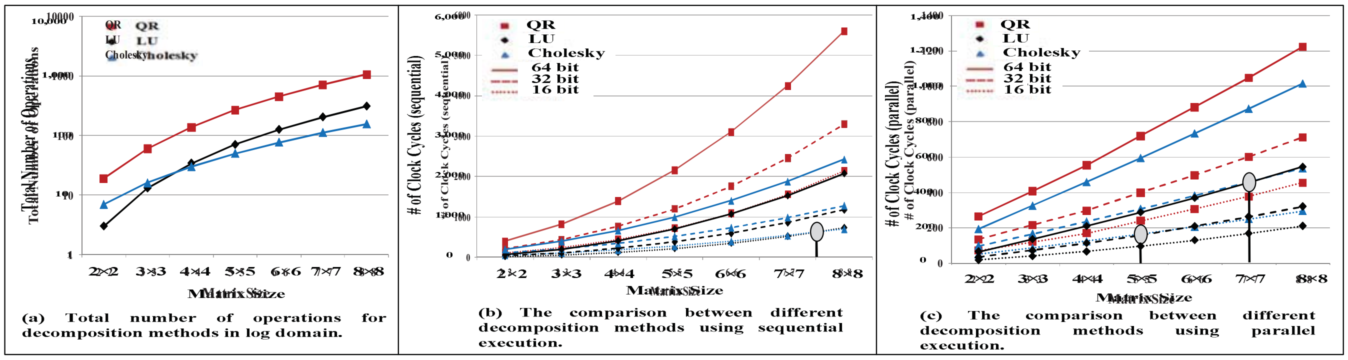


Fig. 6. Different design space exploration: inflection point analyses, of our tool. (a), (b) and (c) show the total number of operations of decomposition methods in log domain, the comparison between different decomposition methods using sequential execution and parallel execution respectively.

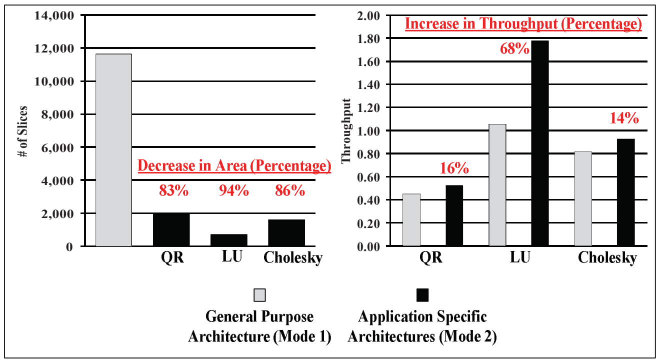


Fig. 7. The comparison of the general purpose architecture (Mode 1 results of our tool) and application specific architectures (Mode 2 results of our tool) in terms of slices (area) and throughput (performance).

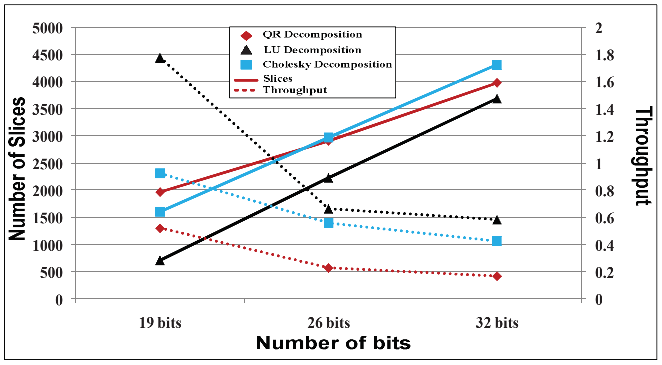


Fig. 8. Area and throughput tradeoffs for different bit width of data: 19, 26 and 32 bits. A user can determine the length of the data by the error analysis part of our tool. High precision can be obtained by using more number of bits as data width which comes at the price of larger area and lower throughput.

QR decomposition requires more clock cycles than the other decomposition methods for all bit widths. Execution of 16 bit QR decomposition requires the same number of clock cycles with the 64 bit LU decomposition. Cholesky decomposition requires more clock cycles than LU decomposition because of its square root operation; however the difference between LU and Cholesky becomes smaller for smaller number of bit widths. The parallel execution of decomposition methods (c) shows that LU decomposition performs better than other methods. 64 bit implementation of LU decomposition performs almost the same as the 32 bit Cholesky decomposition and also the 32 bit LU decomposition performs almost the same as the 16 bit implementation of Cholesky decomposition.

Architectural Design Alternatives: We present two different analyses for comparison of decomposition methods in terms of architectural design alternatives in Figure 7 and Figure 8 for 4 × 4 matrices. The general purpose architecture (Mode 1 results of our tool) and application specific architectures (Mode 2 results of our tool) are compared in Figure 7 in terms of slices (area) and throughput (performance). We compare area and throughput results of different bit width implementations of decomposition methods in Figure 8.

The general purpose architecture is able to perform different decomposition methods with a selection input. However, it is possible to provide better area and throughput results by optimizing the general purpose architecture and creating an application specific architecture for a specific decomposition method. As can be seen in Figure 7, by creating an application specific architecture which uses the optimal number of resources (using Mode 2 of our tool), we can decrease the area by 83%, 94% and 86% for QR, LU and Cholesky decompositions respectively. Optimizing the architecture also provides higher clock frequencies and leads 16%, 68% and 14% increase in throughput for QR, LU and Cholesky decompositions respectively.

We also present area and throughput results for different bit widths of data: 19, 26 and 32 bits in Figure 8. The user can determine the length of the data by the error analysis part of our tool. High precision can be obtained by using a larger number of bits but this comes at the price of larger area and lower throughput. As can be seen in Figure 8, LU decomposition provides the smallest area and highest throughput for all bit widths. Also, it is important to see that there is an inflection point between QR and Cholesky decompositions at 25 bits in terms of area where Cholesky decomposition requires more area for bit widths larger than 25 bits. On the other hand, Cholesky decomposition provides higher throughput compared to QR decomposition for all bit widths due to the fact that Cholesky decomposition requires less clock cycles.

In the next subsection, we present a case study: *Implementation of Adaptive Weight Calculation Core*, using the QR decomposition core generated by GUSTO.

A. Case Study: Implementation of Adaptive Weight Calculation Core using QRD-RLS Algorithm

Adaptive weight calculation (AWC) is required in many wireless communication applications including adaptive beamforming, equalization, predistortion and multiple-input multiple-output (MIMO) systems [13]. These applications involve solving systems of equations in many cases which can be seen as:

$$\begin{matrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = b_1 + e_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m = b_2 + e_2 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m = b_n + e_n \end{matrix}$$

where A is the observations matrix which is assumed to be noisy, b is a known training sequence and x is the vector to be computed. This is described more compactly in matrix notation as $Ax = b + e$. If there is the same number of equations as there are unknowns, i.e. $n = m$, this system of equations has a unique solution, $x = A^{-1}b$. However, high sampling rate communication applications are often over-determined, i.e. $n > m$. Introducing the least squares approach helps to solve the problem by minimizing the residuals: $min \sum_n e_n^2$.

In general, the least squares approach, e.g. Least Mean Squares (LMS), Normalized LMS (NLMS) and Recursive Least Squares (RLS), is used to find an approximate solution to these kinds of system of equations. Among them, RLS is most commonly used due to its good numerical properties and fast convergence rate [14]. However, it requires matrix inversion which is not efficient in terms of precision and hardware implementation. Applying QR decomposition to perform adaptive weight calculation based on RLS is a better method and leads to more accurate results and efficient architectures. Therefore, we use our tool, GUSTO, to implement a QR decomposition core for use in adaptive weight calculation. The resulting upper triangular matrix, R, which is the solution of QR decomposition, is used to find coefficients of the system by back-substitution after converting b into another column matrix, c, such that $Rx = c$.

Comparison: We provide comparisons in Table I using different applications which use decomposition methods. These related works are hard to compare with each other since two of them are matrix inversion architectures [15, 16], one is a beamformer design [17], and ours is an adaptive weight calculation design. We also use fixed point arithmetic and fully utilize FPGA resources like DSP48 multipliers instead of Look-up Tables (LUTs). Therefore our intention is not to directly compare these different designs, but to give an idea about our area and throughput results compared to other implementations that use decomposition methods. Edman et al. proposed a linear array architecture for inversion of complex valued matrices [15]. Karkooti et al. presented an implementation of matrix inversion using the QRD-RLS algorithm along with square GR and folded systolic arrays [16]. Dick et al. considered the architecture, design flow and the verification process of a real-time beamformer [17] which is most similar to our work since area and timing results are presented for QR decomposition and back substitution architectures (clock frequency is assumed as 250 MHz). The advantage of our work compared to the related work is that we give the ability to the designer to study the tradeoffs between architectures with different design parameters to find an optimal design.

V. Conclusion

This paper presents automatic generation and optimization of decomposition methods using a core generator tool, GUSTO, which enables easy design space exploration. GUSTO offers different parameterization options such as resource allocation, bit widths of the data, number of functional units and organization of controllers and interconnects. Therefore, a designer can easily study the area and throughput tradeoffs of different architectures. GUSTO's optimized application specific architectures decrease the area by 83%, 94% and 86% and increase the throughput 16%, 68% and 14% compared to the general purpose architecture for QR, LU and Cholesky decompositions respectively. In this paper, we concentrate on small matrix sizes, and employ fixed point arithmetic in our architectures. Our future work will include solutions for decomposition of larger matrix sizes and usage of floating point arithmetic.

TABLE I
Comparisons Between Our Results and Previously Published Papers. NR Denotes Not Reported.

	Ref[15]	*Ref[16]*	*Ref[17]*		***GUSTO***
Application	Matrix Inversion	Matrix Inversion	Beamformer		**AWC**
Method	QR	QR	QR		**QR**
Matrix Size	4×4	4×4	3×3	5×5	$\mathbf{4 \times 4}$
Bit width	12	20	18		**20**
Data type	fixed	floating	NR		**fixed**
Device type (Virtex)	II	IV	IV		**IV**
Slices	4400	9117	3530		**2558**
DSP48s	NR	22	13		**12**
BRAMs	NR	9	6		**1**
Throughput *($10^6 \times s^{-1}$)*	0.28	0.12	0.27	0.11	**0.13**

References

[1] K. Kusume, M. Joham, W. Utschick, G. Bauch, "Efficient Tomlinson-Harashima precoding for spatial multiplexing on flat MIMO channel,"*IEEE International Conference on Communications*, Volume 3, 16-20 May 2005 Page(s):2021 - 2025 Vol. 3.

[2] L. Zhou, L. Qiu, J. Zhu, "A novel adaptive equalization algorithm for MIMO communication system," *Vehicular Technology Conference*, Volume 4, 25-28 Sept., 2005 Page(s):2408 – 2412.

[3] C. Hangjun, D. Xinmin, A. Haimovich, "Layered turbo space-time coded MIMO-OFDM systems for time varying channels,"*Global Telecommunications Conference*, IEEE Volume 4, 1-5 Dec. 2003 Page(s):1831 - 1836 vol.4.

[4] Y. Meng, A.P.Brown, R. A.Iltis, T. Sherwood, H. Lee, R.Kastner, "MP core: algorithm and design techniques for efficient channel estimation in wireless applications," *Design Automation Conference Proceedings,* 42nd , vol., no., pp. 297-302, 13-17 June 2005.

[5] R. A. Iltis, S. Mirzaei, R. Kastner, R. E. Cagley, B. T. Weals, "Carrier Offset and Channel Estimation for Cooperative MIMO Sensor Networks," *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE* , vol., no., pp.1-5, Nov. 2006.

[6] R.E. Cagley, B. T. Weals, S. A. McNally, R. A. Iltis, S.Mirzaei, R. Kastner, "Implementation of the Alamouti OSTBC to a Distributed Set of Single-Antenna Wireless Nodes," *Wireless Communications and Networking Conference, 2007. WCNC 2007. IEEE*, vol., no., pp.577-581, 11-15 March 2007.

[7] A. Irturk, B. Benson, S. Mirzaei and R. Kastner, "An FPGA Design Space Exploration Tool for Matrix Inversion Architectures," *In Proceedings of IEEE Symposium on Application Specific Processors (SASP) 2008*.

[8] A. Irturk, B. Benson, and R. Kastner, "Automatic Generation of Decomposition based Matrix Inversion Architectures," *In Proceedings of International Conference on Field-Programmable Technology (ICFPT) 2008*.

[9] G.H. Golub, C.F.V. Loan, Matrix Computations, 3rd ed. Baltimore, MD: *John Hopkins University Press*.

[10] A. Bjőrck, C. Paige, "Loss and recapture of orthogonality in the modified Gram-Schmidt algorithm," *SIAM J. Matrix Anal. Appl*., vol. 13 (1), pp 176-190, 1992.

[11] A. Bjőrck, "Numerics of Gram-Schmidt orthogonalization," *Linear Algebra and Its Applications*, vol. 198, pp. 297-316, 1994.

[12] C. K. Singh, S.H. Prasad, P.T. Balsara, "VLSI Architecture for Matrix Inversion using Modified Gram-Schmidt based QR Decomposition", *20th International Conference on VLSI Design.* (2007) 836 – 841.

[13] Simon Haykin, Adaptive Filter Theory, *Prentice Hall*, Fourth Edition.

[14] J. Ma, K.K. Parhi, E.F. Deprettere, "Annihilation-reordering lookahead pipelined CORDIC-based RLS adaptive filters and their application to adaptive beamforming," *IEEE Transactions on Signal Processing*, 2000.

[15] F. Edman, V. Öwall, "A Scalable Pipelined Complex Valued Matrix Inversion Architecture," *IEEE International Symposium on Circuits and Systems*. (2005) 4489 – 4492.

[16] M. Karkooti, J.R. Cavallaro, C. Dick, "FPGA Implementation of Matrix Inversion Using QRD-RLS Algorithm," *Thirty-Ninth Asilomar Conference on Signals, Systems and Computers* (2005) 1625 – 162.

[17] C. Dick, F. Harris, M. Pajic, D. Vuletic, "Real-Time QRD-Based Beamforming on an FPGA Platform," *Fortieth Asilomar Conference on Signals, Systems and Computers*, 2006. ACSSC '06.