An Optimization Methodology for Matrix Computation Architectures

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

†Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093
{airturk, b1benson, kastner}@cs.ucsd.edu

‡Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095
nlaptev@cs.ucla.edu

Abstract— **Matrix computations such as matrix decomposition and inversion are essential for various algorithms which are employed in wireless communication. FPGAs are ideal platforms for such applications; however, the need for vast amounts of customization throughout the design process of a matrix computation core can overwhelm the designer. This paper presents an automatic generation and optimization methodology for different matrix computation architectures using a generator tool, GUSTO, that we developed to enable easy design space exploration with different parameterization options. We especially concentrate on wireless communication MIMO-OFDM applications which often use small matrix dimensions. We present automatic generation of a variety of general purpose matrix computation architectures and optimized application specific architectures. GUSTO's application specific architectures have comparable results to published architectural implementations, but offer the advantage of providing the designer the ability to study the tradeoffs between architectures with different design parameters.**

I. Introduction

Matrix computations lie at the heart of most scientific computational tasks. Matrix computations, matrix decomposition and inversion, are frequently used to solve linear systems of equations in many fields such as wireless communication. For example, in wireless communication, MIMO-OFDM systems use matrix inversion in equalization algorithms to remove the effect of the channel on the signal [1], minimum mean square error algorithms for pre-coding in spatial multiplexing [2] and detection-estimation algorithms in space-time coding [3]. These systems often use a small number of antennas (2 to 8) which results in small matrices to be decomposed and/or inverted. For example the 802.11n standard [4] specifies a maximum of 4 antennas on the transmit/receive sides and the 802.16 [5] standard specifies a maximum of 16 antennas at a base station and 2 antennas at a remote station.

The choice of a computing platform plays an important role in the overall design of these systems. Previously, designers decided between a hardware or software implementation. The hardware implementation consisted of designing an ASIC, which offers exceptional performance, but long time to market and high costs for all but the largest production chips. The software route tended towards the use of DSPs due to the ease of development and fast time to market. However, they lack the performance for high throughput applications. Recently FPGAs have become prevalent for these applications. FPGAs play a middle role between ASICs and DSPs, as they have the programmability of software with performance approaching that of a custom hardware implementation. However, FPGAs require vast amounts of customization throughout the design process and few tools exist which can aid the designer with the many system, architectural and logic design choices. Designing a high level tool for fast prototyping matrix computation architectures is crucial.

For automatic generation and optimization of matrix computation architectures, we designed an easy to use tool, GUSTO (**"G**eneral architecture design **U**tility and **S**ynthesis **T**ool for **O**ptimization"). GUSTO is the first tool of its kind to provide automatic generation and optimization of a variety of general purpose matrix computation architectures with different parameterization options. It also optimizes the general purpose architecture to improve its area results and design quality which results in a scheduled, static, application specific architecture. GUSTO allows the user to select the matrix operation, the matrix dimension, the type and number of arithmetic resources, the data representation (the integer and fractional bit width), and the different modes of operation for general purpose or application specific architectures.

To demonstrate the effectiveness of our methodology, we implement different decomposition methods: QR, LU and Cholesky and different matrix inversion methods: upper triangular matrix inversion, general matrix inversion using QR decomposition, nonsingular diagonally dominant square matrix inversion using LU decomposition and positive definite square matrix inversion using Cholesky decomposition.

Our major contributions are:

- Automatic generation and optimization of different matrix computation architectures with parameterized matrix dimensions, bit widths, resource allocation, modes and methods.
- Comparison of different matrix decomposition and inversion methods in terms of different matrix dimensions, bit widths and parallelism.
- Detailed study of area, timing and throughput tradeoffs of matrix inversion architectures using different parameterizations.

The rest of the paper is organized as follows. In section II, we introduce decomposition methods: QR, LU and Cholesky; and matrix inversion methods. In section III, we introduce our tool and describe the optimizations performed: static architecture generation and trimming for optimization. Section IV presents our implementation results in terms of area and throughput and compares our results with previously

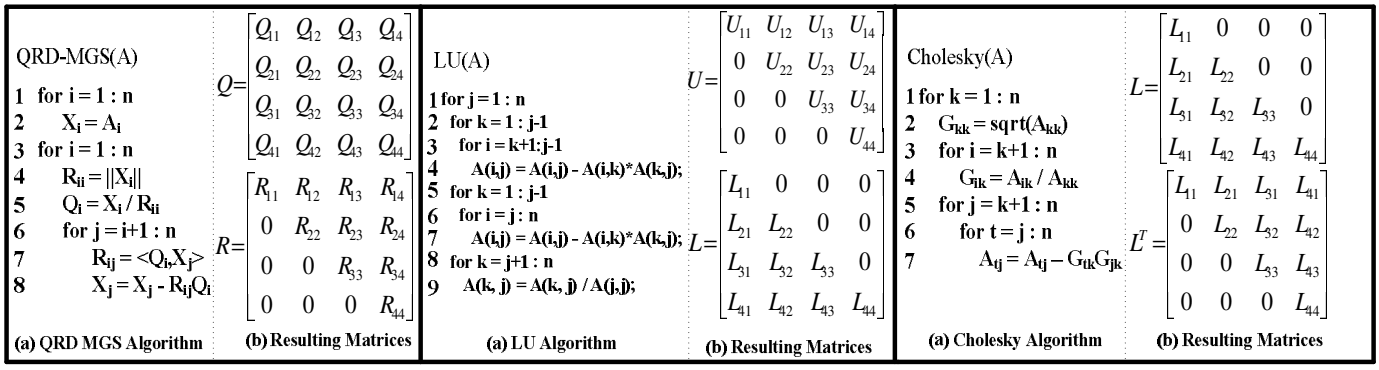

Fig. 1. QR decomposition (QR-MGS) algorithm is presented in (a). The resulting matrices of the decomposition are shown in (b).

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

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

published work. We conclude in Section V.

II. Matrix Decomposition and Inversion Methods

Explicit matrix computation of full matrices is computationally intensive. In many different computations such as matrix inversion, one should consider converting this problem into an easy decomposition problem which will result in analytic simplicity and computational convenience. Below we describe three known and widely used decomposition methods: QR, LU and Cholesky decomposition methods [6]. For square matrices, n denotes the matrix size of the matrix such that $n = 4$ for 4×4 matrices. For rectangular matrices, m and n denote the number of rows and columns in the matrix respectively such that $m = 3$, $n = 4$ for 3×4 matrices.

- **QR:** Given $A \in \mathbb{R}^{m \times n}$ with $rank(A) = n$, QR factorization exists as $A = Q \times R$ where $Q \in \mathbb{R}^{m \times n}$ has orthonormal columns and $R \in \mathbb{R}^{n \times n}$ is upper triangular.
- **LU:** Given $A \in \mathbb{R}^{n \times n}$ with $det(A(1:k, 1:k)) \neq 0$ for $k = 1 : n-1$, LU decomposition exists as $A = LU$. If LU decomposition exists and the given matrix A, is nonsingular, then the decomposition is unique and $det(A) = u_{11} \ldots u_{nn}$.
- **Cholesky:** Given a symmetric positive definite matrix, $A \in \mathbb{R}^{n \times n}$, Cholesky decomposition exists as $A = G \times G^T$ where $G \in \mathbb{R}^{n \times n}$ is a unique lower triangular matrix with positive diagonal entries.

where a matrix $A \in \mathbb{R}^{n \times n}$ is *positive definite* if $x^T A x > 0$ for $x \in \mathbb{R}^n$ and $x \neq 0$ where if A is *symmetric positive definite matrix* then $A^T = A$. A positive definite matrix is always nonsingular and its determinant is always positive.

Cholesky and LU decompositions work only with positive definite and nonsingular diagonally dominant square matrices, respectively. On the other hand, QR decomposition is more general and can be applied to any matrix. We further explain these decomposition methods, their characteristics and algorithms, the resulting matrices and the solution steps for matrix inversion in the next subsections.

A. QR Decomposition

QR decomposition is an elementary operation, which decomposes a matrix into an orthogonal and a triangular matrix. QR decomposition of a matrix A is shown as $A = Q \times R$, where Q is an orthogonal matrix, $Q^T \times Q = Q \times Q^T = I$, $Q^{-1} = Q^T$, and R 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. Applying slight modifications to the Classical Gram-Schmidt (CGS) algorithm gives the Modified Gram-Schmidt (MGS) algorithm [6]. QRD-MGS is numerically more accurate and stable than QRD-CGS and it is numerically equivalent to the Givens Rotations solution [7] (the solution that has been the focus of previously published hardware implementations because of its stability and accuracy). Also, 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 is accurate to floating-point machine precision [7]. 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 and its leading principal submatrices are all nonsingular, 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

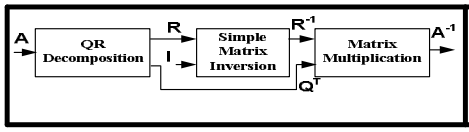

Fig. 5. The solution steps of the general matrix inversion.

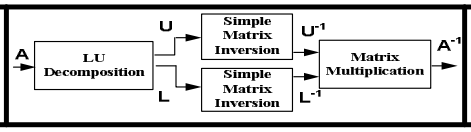

Fig. 6. The solution steps of the nonsingular diagonally dominant square matrix inversion.

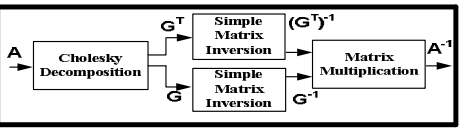

Fig. 7. The solution steps of the positive definite square matrix inversion.

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 into a unique lower triangular matrix with positive diagonal entries. 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), second column: A_{22}, A_{32}, A_{42}, third column: A_{33}, A_{43}, and fourth column: A_{44} are updated by (7).

Decomposition methods such as QR, LU and Cholesky, provide a means to simplify the matrix inversion. The selection of the decomposition method depends on the characteristics of the given matrix. For non-square matrices or when simple inversion to recover the data performs poorly, the QR decomposition is used to generate an equivalent upper triangular system. For simpler detection via inversion of square channel matrices, the LU and Cholesky decompositions are compatible with positive definite and nonsingular diagonally dominant square matrices, respectively. We introduce these matrix inversion methods in the following subsections.

D. Matrix Inversion of Triangular Matrices

Triangular matrix inversion is used in all of the decomposition based (QR, LU and Cholesky) matrix inversion

```
    R-1 = 0
    for j = 1 : n
        for i = 1 : j-1
            for k = 1 : j-1
1               R-1ij = R-1ij + R-1ikRkj
        for k = 1 : j-1
2           R-1kj = -R-1kj / Rjj
3   R-1jj = 1 / Rjj
```

Fig. 4. Matrix Inversion of upper triangular matrices.

architectures described above and we use this subsection to describe why this inversion is relatively simple and therefore not a dominant calculation in any of these methods. Primarily, triangular matrix inversion requires fewer calculations compared to full matrix inversion because of its zero entries. The algorithm for triangular matrix inversion is shown in Figure 4 and described below.

Upper triangular matrix inversion is performed column by column. Calculating the diagonal entries of the R^{-1} matrix consists of simply dividing 1 by the diagonal entry of the R matrix (3) and the rest of the column entries introduce multiplication and addition iteratively (1) which is then divided by the diagonal R matrix entry (2).

E. General Matrix Inversion

General matrix inversion which is applicable to any matrix dimension and characteristic employs QR decomposition. The solution for the inversion of a matrix A^{-1}, using QR decomposition is shown as $A^{-1} = R^{-1} \times Q^T$. This solution consists of three different parts: QR decomposition, matrix inversion for the upper triangular matrix and matrix multiplication which is shown in Figure 5.

F. Nonsingular Diagonally Dominant Matrix Inversion

Nonsingular diagonally dominant square matrix inversion employs LU decomposition. The solution for the inversion of a matrix A^{-1}, using LU decomposition is shown as $A^{-1} = U^{-1} \times L^{-1}$. This solution consists of four different parts: LU decomposition of the given matrix, matrix inversion for the lower triangular matrix, matrix inversion of the upper triangular matrix and matrix multiplication which is shown in Figure 6.

G. Positive Definite Matrix Inversion

Positive definite square matrix inversion employs Cholesky decomposition. The solution for the inversion of a matrix A^{-1}, using Cholesky decomposition is shown as $A^{-1} = (G^T)^{-1} \times G^{-1}$. This solution consists of four different parts: Cholesky decomposition, matrix inversion for the transpose of the lower triangular matrix, matrix inversion of the lower triangular matrix and matrix multiplication which is shown in Figure 7.

III. Matrix Computation Core Generator Tool

There are many architectural design choices while implementing the hardware for different matrix computations. These implementation choices are: matrix operation and its method (depends on the structure of the given matrices), matrix size (depends on the number of antennas used in MIMO-OFDM systems), resource allocation, number of functional units, the organization of controllers and interconnects (depends on the hardware constraints such that designs which offer the most time efficient or the most area

efficient architecture), and bit widths of the data (depends on the precision required). Not only is generating the hardware for given requirements tedious work, but performing a design space exploration to find the optimum hardware is a time consuming processes. Therefore, a high level tool for design space exploration and fast prototyping is essential and required.

GUSTO, "**G**eneral architecture design **U**tility and **S**ynthesis **T**ool for **O**ptimization," is such a high level design tool, written in Matlab, that is the first of its kind to provide design space exploration across different matrix computation architectures. GUSTO allows the user to select the matrix operation and its method, the matrix dimension, the type and number of arithmetic resources, the data representation (the integer and fractional bit width), and two modes of operation (Mode 1 or Mode 2) as shown in Figure 8. GUSTO also performs error analysis after the resource allocation step to find an appropriate fixed point representation which provides results with the accuracy similar to that of a floating point implementation. GUSTO takes the sample input data which is generated by the user. The matrix computation is performed using single or double precision floating point arithmetic and these are referred as the actual results. The same calculations are performed using different bit widths of fixed point representations to determine the error, the difference between the actual and the computed result. GUSTO provides three different metrics to the user to determine if the accuracy is enough for the application: mean error, standard deviation of error, and mean percentage error.

GUSTO has two different modes of operation. Mode 1 provides a general purpose architecture while Mode 2 provides an application specific architecture. The general purpose architecture is used for area and timing analysis of a general non-optimized solution, and the advantage of this architecture is that it is capable of solving different matrix computations with a selection input. Unfortunately, Mode 1's general purpose architectures generally do not lead to high-performance results. When the user knows the environmental requirements and matrix characteristics which will be encountered, choosing a specific method and creating an application specific architecture by optimizing/customizing these architectures to improve their area results is another essential step to enhance design quality.

In Mode 2, GUSTO creates a scheduled, static, application specific architecture while ensuring the correctness of the solution is maintained. We divided these optimizations into two sections: static architecture generation and trimming for optimization.

Static architecture generation: Mode 1 of GUSTO generates a general purpose architecture and its datapath by using resource constrained list scheduling after the required inputs are given. Simulating this architecture in Mode 2 helps us to reveal the assignments done to the arithmetic units and the memory elements during the scheduling process. Gathering this information and using it to cancel the scheduling process and dynamic memory assignments results in a static architecture with significant area and timing savings.

Trimming for optimization: GUSTO performs trimming/removing the unused resources from the general purpose architecture while ensuring that correctness of the solution is maintained. GUSTO simulates the architecture to define the usage of arithmetic units, multiplexers, register entries and input/output ports and trims away the unused components with their interconnects. A trimming example is shown in Figure 9. Suppose there are 2 arithmetic units with 2 inputs/1 output each and one memory with 1 input/2 outputs (a). Input / output port relationships between arithmetic unit *A* and the other units are shown in a block diagram in (b). Although *Out_A*, *Out_B*, *Out_mem1*, and *Out_mem2* are all inputs to *In_A1* and *In_A2*, not all the inputs may be used during computation. We can represent whether an input/output port is used or not during simulation in a matrix such as the one shown in (c). As the simulation runs, the matrix is filled with 1s and 0s representing the used and unused ports respectively. GUSTO uses these matrices to remove the unused resources (d). In this example, two inputs, *Out_A*, *Out_mem1* to *In_A1* and another two inputs, *Out_B*, *Out_mem2* to *In_A2* are removed.

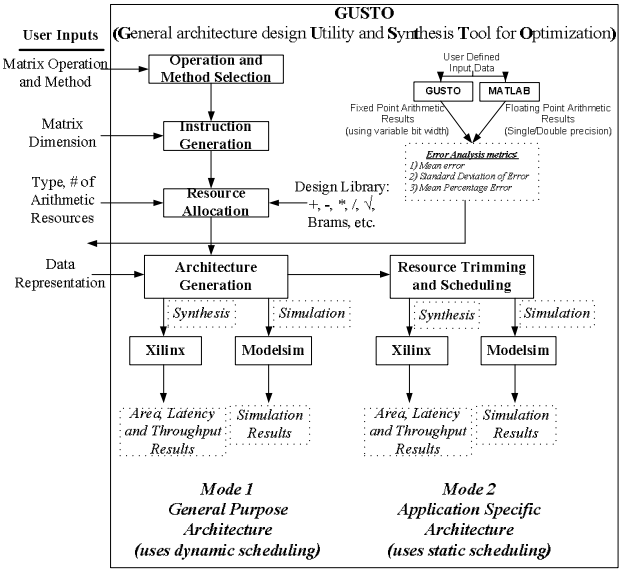

Fig. 8. Different modes of GUSTO.

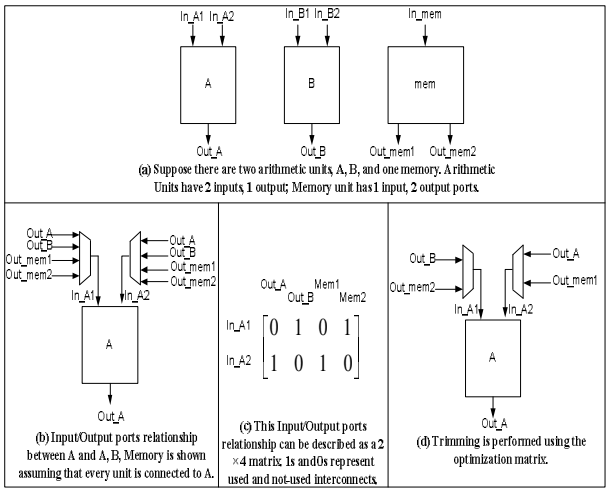

Fig. 9. Flow of GUSTO's trimming feature.

IV. RESULTS

In this section, we present different design space exploration examples using different inputs of GUSTO and compare our results with previously published FPGA implementations. Design space exploration can be divided into two parts, inflection point analysis and architectural design alternatives analysis.

Inflection Point Analysis: We partitioned inflection point analysis into another two parts. First, we present the results for decomposition methods which help us to quantify their effects on the computation of matrix inversion and then we present matrix inversion results. We present different execution results, serial and parallel, for different bit widths and matrix dimensions to answer: at what matrix size does an inflection point occur and how does varying bit width and degree of parallelism change the inflection point? The comparisons for sequential and parallel executions of matrix decomposition and inversion methods are shown in Figure 10 (a, b, c, d) with different bit widths: 16, 32 and 64. Square, spade and triangle represent QR, LU and Cholesky methods respectively. Solid, dashed and smaller dashed lines represent 64, 32 and 16 bits of bit widths respectively. The balloons denote the inflection points between these methods for the different bit widths.

The sequential execution results of decomposition methods (a) show that the *QR decomposition* method executes a significantly higher number of clock cycles than the other methods. The 16 bit *QR decomposition* implementation requires the same number of clock cycles with the 64 bit *LU decomposition* implementation. *Cholesky decomposition* takes more clock cycles than *LU decomposition* where this difference becomes smaller for smaller number of bit widths. The sequential execution results of matrix inversion (c) show that *general matrix inversion* takes more clock cycles than *positive definite matrix inversion* and *nonsingular diagonally dominant square matrix inversion* again where *positive definite matrix inversion* takes more cycles than *nonsingular diagonally dominant square matrix inversion.* As the bit widths get smaller, the difference between *general matrix inversion* and the other methods does not change, however the difference between *positive definite matrix inversion* and *nonsingular diagonally dominant square matrix inversion* becomes smaller.

The parallel execution results of decomposition methods (b) show that *QR decomposition* and *Cholesky decomposition* get closer to each other where *LU decomposition* performs better than the others. It is important to see that the 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.* The parallel execution results of matrix inversion (d) show that *general matrix inversion* have the highest number of clock cycles for all bit widths where *positive definite matrix inversion* and *nonsingular diagonally dominant square matrix inversion* have a similar number of clock cycles for small bit widths. However, *nonsingular diagonally dominant square matrix inversion* uses increasingly fewer clock cycles than *positive definite matrix inversion* with increasing bit widths and matrix dimensions. *Nonsingular diagonally dominant square matrix inversion* with 32 bits performs almost the same as *general matrix inversion* with 16 bits. Also, the 64 bits *nonsingular diagonally dominant square matrix inversion* performs almost the same as the 32 bits *general matrix inversion* in terms of total number of clock cycles.

Architectural Design Alternatives: These analyses are shown for matrix inversion for different bit widths and matrix sizes. We present area results in terms of slices and performance results in terms of throughput. Throughput is calculated by dividing the maximum clock frequency (MHz) by the number of clock cycles to perform matrix inversion. All 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. Both mode 1 (non-optimized) and mode 2 (optimized) results are shown for *general matrix inversion* in Figure 11 (a) to show the improvement in the results with the optimization feature. It is shown that area and throughput increase up to the optimal number of resources as the number of resources increase. However, adding more than the optimal number of resources decreases throughput while still increasing area. Mode 2 of GUSTO finds the optimal number of resources which maximizes the throughput while minimizing area where the application specific architecture provides an average of 59% decrease in area and 3X increase in throughput over

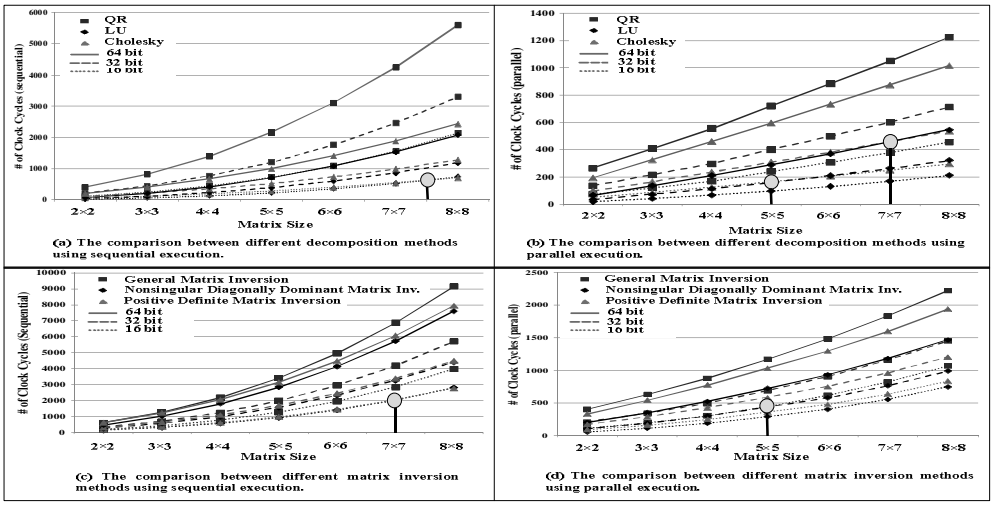

Fig. 10. Different design space exploration: inflection point analyses, of our tool. Top row: decomposition methods only. Bottom row: matrix inversion.

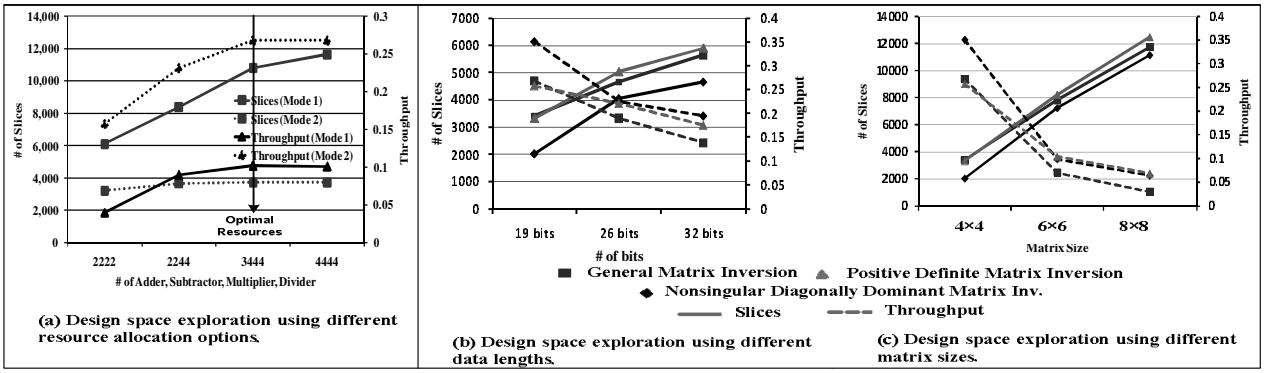

Fig. 11. Different design space exploration examples, specifically area and throughput results of different bit width and matrix dimensions, of our tool.

Mode 1's general purpose (non optimized) design.

Bit width of the data is another important input for the matrix inversion. The precision of the results is directly dependent on the number of bits used. The usage of a high number of bits results in high precision at a cost of higher area and lower throughput. We present 3 different bit widths, 19, 26 and 32 bits in (b) for these three different matrix inversion architectures. Usage of *nonsingular diagonally dominant square matrix inversion* results in smallest area and highest throughput compared to the other methods. *Positive definite matrix inversion* offers higher throughput at a cost of larger area compared to *general matrix inversion*.

We also present three different matrix dimension, 4×4, 6×6 and 8×8, implementation results in (c) showing how the area and performance results scale with matrix dimension. We again observe that *nonsingular diagonally dominant square matrix inversion* offer better area and throughput results compared to other methods for all matrix sizes.

Comparison: We provide a comparison between our results and previously published implementations for 4×4 matrices in Table 1. We present all of our implementations with bit width 20 as this is the largest bit width value used in the related works. Though it is difficult to make direct comparisons between our designs and those of the related works (because we used fixed point arithmetic instead of floating point arithmetic and fully used FPGA resources (like DSP48s) instead of LUTs), we observe that our results are comparable. The main advantages of our implementation are that it provides the designer the ability to study the tradeoffs between architectures with different design parameters and provides a means to find an optimal design.

V. Conclusion

This paper presents matrix computation architectures using a generator tool, GUSTO, that is developed to enable easy design space exploration. GUSTO provides different parameterization options including matrix dimensions, bit width and resource allocations which enable us to study area and performance tradeoffs over a large number of different architectures. In this paper, we especially concentrate on matrix decomposition and inversion methods for wireless communication, MIMO-OFDM systems which often use small matrices, to observe the advantages and disadvantages of these methods in response to varying parameters. GUSTO is the only tool that allows design space exploration across different matrix computation architectures.

TABLE I
COMPARISONS BETWEEN OUR RESULTS AND PREVIOUSLY PUBLISHED PAPERS. NR DENOTES NOT REPORTED.

	Ref[8]	*Ref[9]*	***Our***	***Our***	***Our***
Method	QR	QR	**QR**	**LU**	**Cholesky**
Bit width	12	20	**20**	**20**	**20**
Data type	fixed	floating	**fixed**	**fixed**	**fixed**
Device type (Virtex)	II	IV	**IV**	**IV**	**IV**
Slices	4400	9117	**3584**	**2719**	**3682**
DSP48s	NR	22	**12**	**12**	**12**
BRAMs	NR	NR	**1**	**1**	**1**
Throughput *($10^6 \times s^{-1}$)*	0.28	0.12	**0.26**	**0.33**	**0.25**

We would like to give information about our previous work since the paper submission is blind review. In our previous work, we compare two different matrix inversion methods: QR decomposition based and analytic method since they are applicable to any matrix structure. We determine different inflection points and present their architectural results. In this work, we present a methodology to handle any matrix computation and compare decomposition methods: QR, LU and Cholesky, matrix inversion methods: general matrix inversion, nonsingular diagonally dominant matrix inversion and positive definite matrix inversion. This discussion will be added to our paper if our paper gets accepted.

References

[1] 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.

[2] 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.

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

[4] "IEEE 802.11 LAN/MAN Wireless LANS," IEEE *Standards Association*, http://standards.ieee.org/getieee802/802.11.html.

[5] "IEEE 802.16 LAN/MAN Broadband Wireless LANS," IEEE *Standards Association*, http://standards.ieee.org/getieee802/802.16.html.

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

[7] 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.

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

[9] 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.