Design and Implementation of High Performance Matrix Inversion Based on Reconfigurable Processor

Kun Wang, Li Li\textsuperscript{a)}, Feng Han, Fan Feng, Jun Lin
School of Electronic Science and Engineering, Nanjing University
Nanjing 210023, China

\textsuperscript{a)} lili@nju.edu.cn

Abstract: In this paper, we propose a high performance matrix inversion implementation on a reconfigurable application specific processor. Our implementation can accelerate variable order matrix inversion ranging from 4 to 144. We adopt LU decomposition to reduce the computation complexity and a pivoting operation to ensure the stability. In order to get higher performance within the limited resources, parallel computing and time-sharing multiplexing are employed. The chip testing results show that our implementation improve the performance of inversion efficiently. The highest parallel speed-up ratio can achieve 3 times, and the execution time of a 144×144 matrix inversion is 4.07ms.

Keywords: reconfigurable processor, matrix inversion, LU decomposition, parallel computing, time-sharing multiplexing

Classification: Integrated circuits

References

1 Introduction

With the requirements for higher performance and flexibility in embedded design, reconfigurable computing [1, 2, 3] is becoming more promising than application specific integrated circuit (ASIC) and digital signal processor (DSP). Reconfigurable architectures have been proposed as a compromise approach as they are flexible, scalable and can provide reasonable computing capability. A domain-specific reconfigurable architecture should support multiple algorithms so that it can provide various combinations of functionalities and parameters. In the field of radar signal processing, it should achieve the required calculating performance for computationally intensive algorithms, such as finite impulse response (FIR), fast Fourier transform (FFT), correlation, matrix operation, etc. However, we rarely find reconfigurable processors to accelerate the executing of large scale matrix inversion.

Matrix inversion is one of the most computationally intensive linear alge-
bra algorithms that has drawn attention for decades. The complexity of the direct analytic method increases exponentially with the size of matrix, so matrix decomposition becomes the most common method applied in matrix inversion with large dimensions, such as LU decomposition (with partial pivoting) [4, 5, 6, 7], QR decomposition [8, 9, 10], Cholesky decomposition [11], etc. By analyzing these algorithms [12], LU decomposition (with partial pivoting) has better generality than Cholesky decomposition which only applies to symmetric positive definite matrices, and lower computational complexity than QR decomposition. Therefore, we adopt LU decomposition (with partial pivoting) in this paper.

Nowadays, a number of hardware implementations for matrix inversion have been presented based on field programmable gate array (FPGA) and DSP. FPGA can achieve a high throughout, but lacks of certain flexibility. Through literature research, we find that many FPGA-based [13, 14, 15] matrix inversion are realized for matrices of specific size. Moreover, DSP can accomplish matrix inversion by software programming which is more flexible than FPGA. But the features of DSP limit its use in the real-time applications where parallelism and high throughout are required. For these reasons, reconfigurable architecture is a balanced solution for matrix inversion instead of those traditional solutions.

In this paper, we present a hardware implementation of matrix inversion based on a reconfigurable application specific processor (RASP). The proposed implementation supports a range of matrix dimensions from 4 to 144 and reduces the computation complexity of matrix inversion by adopting LU decomposition with partial pivoting. The whole processing of matrix inversion is divided into three parts, namely, LU decomposition, triangular matrix inversion and matrix multiplication. In order to save the hardware resources and improve the performance, we apply time-sharing multiplexing and parallel computing in the design phase. An prototype chip is designed and fabricated on 40nm complementary metal-oxide-semiconductor (CMOS) technology. The testing results show that the maximum speed-up ratio achieves 3 compared with the serial implementation. For a 144×144 complex valued matrix, the execution time is 4.07ms at the frequency of 500MHz. Furthermore, this processor can be reused for many other algorithms such as complex matrix multiplication, filtering, correlation and FFT/IFFT.

The rest of the paper is organized as follows: Section 2 introduces the mathematical principles of matrix inversion based on LU decomposition. Section 3 describes the architecture of RASP. Section 4 describes how to map our design of matrix inversion on RASP. Section 5 discusses the implementation and the experimental results. Section 6 concludes the paper.

2 Backgrounds

2.1 LU decomposition with partial pivoting

The Gaussian elimination based LU decomposition decomposes a $n \times n$ matrix $A$ into a $n \times n$ lower triangular matrix $L$ (the diagonal elements are all 1) and
a $n \times n$ upper triangular matrix $U$ such that $A = LU$. The elements of matrix $L$ and matrix $U$ are computed by the following codes in Algorithm 1. The lower triangular matrix $L$ (except zeros above the diagonal and the diagonal elements) and upper triangular matrix $U$ (except zeros below the diagonal) are stored in the original position of $A$.

**Algorithm 1** LU decomposition with partial pivoting

1: for $k = 1; k < n - 1; k += 1$ do
2:   $piv = \max(A[k : n, k]);$
3:   for $m = 1; m < n; m += 1$ do
4:     $swap(A[piv, m], A[k, m]);$
5:     $swap(P[piv], P[k]);$
6:   end for
7:   for $i = k + 1; i < n; i += 1$ do
9:   end for
10:  for $j = k + 1; j < n; j += 1$ do
11:     for $i = k + 1; i < n; i += 1$ do
13:     end for
14:   end for
15: end for

In Algorithm 1, line 8 is called column normalization and line 12 is called submatrix modification. However, when the absolute value of $A[k, k]$ in column normalization is too small or even zero, it may cause numerical instability as the quotient is too large. In order to avoid the above issue, a pivoting operation should be added before the column normalization to make sure the largest element is on the diagonal of the matrix. The function $\max$ in line 2 is used to find the element of maximum absolute value in vector $A[k : n, k]$ and return the index of the pivot. By increased pivoting, LU decomposition equation becomes $PA = LU$, where $P$ is a permutation matrix. We use a permutation vector to store the permutation matrix. Vector $P$ in Algorithm 1 is used to store the permutation matrix and its initial value is $1, 2, 3, ..., n$.

### 2.2 Triangular matrix inversion

The algorithm of a lower triangular matrix inversion is shown below in Algorithms 2. There are a lot of inherent parallelism which make it suitable for hardware acceleration. Elements in the same row of the result matrix $S$ have no data dependency with each other. Similarly, the upper triangular matrix $U$ has the equation $U^{-1} = ((U^{-1})^H)^H = ((U^H)^{-1})^H$.

### 2.3 Matrix multiplication

According to the formula $A^{-1} = U^{-1}L^{-1}P$, we get the inverse matrix of $A$. $P$ is a permutation matrix which reorders the columns of $U^{-1}L^{-1}$, when right-
Algorithm 2 Triangular matrix inversion

1: for \( i = j = 1; i < n; i ++ \) do
2: \( \text{S}[i, j] = 1/L[i, j]; \)
3: end for
4: for \( i = 2; i < n; i ++ \) do
5: \( \text{for} \ j = 1; j < i - 1; j ++ \) do
6: \( \text{for} \ k = j; k < i - 1; k ++ \) do
7: \( \text{S}[i, j] = \text{S}[i, j] + L[i, k] * \text{S}[k, j]; \)
8: end for
9: \( \text{S}[i, j] = (-1) * \text{S}[i, i] * \text{S}[i, j]; \)
10: end for
11: end for

multiplied to \( U^{-1}L^{-1} \). Matrix multiplication is an iterative procedure and calculates the rows of the result matrix sequentially. There is no correlation between different rows of the result matrix.

3 RASP Architecture

RASP is a reconfigurable computing processor aiming at specific algorithms in the field of radar, such as matrix-vector operations, FFT, FIR, correlation and so on. RASP can speed up the loops of these specific algorithms, so as to meet the high real-time capability for radar signal processing. The interconnection of the basic computing units in RASP can be rearranged through static reconfiguration. Take the tradeoff between performance and silicon area into consideration, the computational resources and the on-chip memory capacity are decided, the detailed structure of RASP will be described below.

![Fig. 1. Overall architecture of RASP](image)

The overall architecture of RASP is shown in Fig. 1, including main controller (MC), reconfiguration controller (RC), reconfigurable computing
array (RCA), direct memory access (DMA), AXI interface and memory. MC is a micro-controller which controls the whole processing flow of RASP. It is mainly composed of a finite state machine and a set of registers. Through these registers, RASP can be configured and queried by external processor which is connected on the AXI bus. The configuration information for RASP includes the algorithm type and size, as well as the source data address and result data address. An application program interface (API) is developed for host processor to schedule RASP. RC is composed of different algorithm controllers including matrix inversion. The data flow between memory and RCA is controlled by these algorithm controllers.

The RCA is composed of 6 heterogeneous reconfigurable processing elements (RPEs), which is responsible for the computing tasks in the system. RPE1-RPE4 have the same structure which contains one complex multiplier, one real multiplier and four complex adders, RPE5 only contains a fixed-point to floating-point converter, and RPE6 contains one complex multiplier, one real multiplier, two real adders and two real dividers. All of these processing elements except real divider are pipelined. According to the requirements of different algorithms, RPEs can be configured to different functions through the control of multiplexer, such as multiplication, addition, multiply-addition, multiply-accumulation, etc. The brief architecture of RPEs are illustrated in Fig. 2 and Fig. 3. There are three main input sources of RPEs: 1) data directly read from the memory; 2) temporary data from register; 3) result data from other RPEs. Accordingly, the output of RPEs can also be saved directly to memory or saved to register or passed to other RPEs. The parallel part of our realization of matrix inversion uses RPE1-RPE4, and the sequential part uses RPE6 (detailed implementation will be given in Section 4).

The total capacity of local memory is 2 mega-byte, which is divided into 32 banks in order to meet the need of data while parallel computing. The data width of each bank module is 64 bits, due to the complex number operation in RASP. DMA can transfer data between the external memory and the local memory via AXI bus.

The procedure of matrix inversion is as follows: Firstly, the necessary configuration information, such as matrix order and addresses of the source data, is written into the configuration register of RASP through AXI interface. If RASP is idle, MC will start the DMA to transfer the source data, and then inform the RC to perform matrix inversion. The process control module of matrix inversion in RC controls the input data and multiplexers of RPEs. When MC receives the finish signal from RC, it will configure DMA to transfer the result matrix and then handle the next configuration information.

4 Implementation of matrix inversion based on RASP

In this section, we present a hardware design for complex matrix inversion based on RASP which can efficiently exploit parallelism and resource reuse.
As described in part 2, the matrix inversion algorithm is composed of LU decomposition, triangular matrix inversion and matrix multiplication. In the same way, our hardware implementation is also divided into three steps which are controlled by a state machine. To efficiently implement matrix inversion on RASP, each step is time-sharing multiplexing use of RPEs. As depicted in section 3, RPEs can be configured to different functions through the control of multiplexer. These multiplexers are managed by the context registers which are under control of the hardware modules, and the reconfiguration can be done in about five cycles. In addition, the data transfer from local memory to RPEs is continuous and the computing in RPEs is executed in pipeline fashion. The following section will describe how to mapping the matrix inversion to RASP.

4.1 Implementation of LU decomposition

Based on the analysis in the previous section, for a $n \times n$ matrix, the process of LU decomposition requires $n - 1$ iterations. It computes a column of $L$ and a row of $U$ at each iteration and uses them to update the trailing submatrix. Our hardware implementation is shown below in Fig. 2.

Step 1: During the $k$th iteration, the elements of column $k$ need to be pivoted. Pivoting is to compare the pivot element $A[k,k]$ with other values.
in this column. Because the processing object is a complex matrix, we use two real multipliers and a real adder to get the sum of squares of the real part and imaginary part of each element in vector \( A[k : n, k] \). Then a data comparator is used to find out the biggest value in the column, as shown in Fig. 2. The MAX block in Fig. 2 is a register which stores the biggest value in the column and will be used in the next step. If the diagonal element is the biggest value in this column, no pivoting is required. If a value found in column \( k \) is bigger than the diagonal element, a pivoting operation has to be performed. The row vector \( A[piv, m](1 \leq m \leq n) \) needs to exchange with the row vector \( A[k, m](1 \leq m \leq n) \) to make sure the correctness of the LU decomposition. The row index of the pivot element is stored in memory and the controller performs the data exchanging operation according to the row index. Thus, a row of matrix \( U \) has been calculated.

Step 2: After the data exchanging operation, the elements \( A[i, k](k+1 \leq i \leq n) \) of column \( k \) need to do the column normalization operation during the \( k \)th iteration. The real divider in RPE6 is non-pipelined, so we use two real dividers to get the reciprocal of the pivot element. Then vector \( A[i, k](k+1 \leq i \leq n) \) is updated by a complex multiplier, as shown in Fig. 2. The reverse block in Fig. 2 is used to reverse the sign bit of the imaginary part of \( A[k, k] \). Thus, a column of matrix \( L \) has been calculated.

Fig. 3. Computing Flow of Step 3 of LU Decomposition

Step 3: During the \( k \)th submatrix modification, the elements of the different columns \( A[i, k+1], A[i, k+1], \ldots, A[i, k+n](k+1 \leq i \leq n) \) are calculated by the
same multiplier \(A[i, k](k + 1 \leq i \leq n)\). There is no data dependencies between different columns, so that submatrix modification can be parallel performed. Thus, RPE1-RPE4 are configured to the same structure as shown in Fig. 3, each of them is composed of a complex multiplier and a complex adder. The red line in Fig. 3 represents the data flow of RPE1-RPE4. These computing units refresh the data of \(A[i, k+1], A[i, k+2], A[i, k+3], A[i, k+4](k+1 \leq i \leq n)\) simultaneously, then keep on doing the submatrix modification operation on the next four columns. Until the last column completes the submatrix modification operation, the new submatrix is generated.

All steps are recursively applied to the new submatrix generated in Step 3. When \(k = n - 1\), the LU decomposition is completed.

### 4.2 Implementation of triangular matrix inversion

The implementation of triangular matrix inversion is also divided into three steps. Matrix \(L\) and the transpose matrix of \(U\) are operated at the same time.

Step 1: According to the codes of lower triangular matrix inversion, the elements in the diagonal line should be divided by one. Similar to the step 1 and step 2 of LU decomposition, RPE6 is configured to complete the reciprocal operation. All the multiplexers in Fig. 2 are selected to 2, thus the comparator and the complex multiplier are bypassed.

---

**Fig. 4.** Computing Flow of Step 2 of Triangular Matrix Inversion
Step 2: This step is to calculate the sum of the products of \(L[i,k]\) and \(S[k,j](1 \leq j \leq n-1, j+1 \leq i \leq n, j \leq k \leq i-1)\), which needs weightiest operation. According to the codes, there is a data dependency between rows of the result matrix, and both original matrix and result matrix should be used. However, elements in the same row are calculated independently, they can be processed concurrently. Therefore, elements should be calculated by row in this step. In this part, RPE1-RP4 are configured to multiply-accumulate unit as shown in Fig. 4, each multiply-accumulate unit consists of one complex multiplier and two complex adders. The inverse process of matrix \(L\) and \(U\) respectively use two multiply-accumulate units at the same time.

Step 3: At last, each element of \(S[i,j](1 \leq j \leq n-1, j+1 \leq i \leq n)\) should reverse its sign bit and multiply the diagonal element \(S[i,i]\) which is processed in step 1. Actually, matrix \(L\) only need to reverse the sign bit of the results of multiply-accumulate units, because its diagonal elements are 1. Among the three steps, step 2 and step 3 are executed in loops. For elements in each row of the matrix, they cannot start the calculation of step 2 until elements in the previous row finished the calculation of step 3.

4.3 Implementation of matrix multiplication
As it is clear from the definition of matrix multiplication that all elements of the result matrix can be computed independently. In this paper, we adopt block matrix multiplication to parallel computing. As shown in Fig. 5, Matrix \(U^{-1}\) is divided into four pieces according to the row, and multiply with matrix \(L^{-1}\) respectively. The basic calculation of matrix multiplication is based on multiply accumulation, which is similar to step 2 of triangular matrix inversion. Hence, RPE1-RPE4 have the same structure illustrated in Fig. 4. All computing units work in pipeline style that guarantees a high processing speed and increases the usage efficiency of resource. At last, the result of \(U^{-1}L^{-1}\) needs column exchanging based on the permutation matrix \(P\).

![Fig. 5. Computing flow of matrix multiplication](image-url)
5 Prototype chip and testing results

A prototype chip of the SoC named as HRMP is implemented based on the 40nm CMOS technology. The size of the RASP is 19.2\text{mm}^2 and the operation frequency is 500MHz. A development board is designed and implemented, RASP can be configured and tested by DSP under VxWorks system. The source data transfer to DDR3 through the Ethernet. Source data is generated by MATLAB on PC, and then delivered to the DDR3 on board through the Ethernet. The result data is transferred back to PC for correctness and precision analysis.

In order to make performance comparison, a serial computing of matrix inversion is also designed. Fig. 6 shows the speed-up ratio of our proposed implementation with serial computing of matrix inversion. The experimental results show that with the increase of matrix size, the implementation based on RASP has a satisfactory speedup compared with the serial computation. The maximum speed-up ratio is close to 3, and this is mainly because the proportion of loop iterations increase with the size of matrix. The detailed time consumed are shown in Table I, the matrix dimension varies from 16 to 144 and the entire execution procedure consists of computation and data transmission.

![Fig. 6. Parallel speed-up ratio](image)

<table>
<thead>
<tr>
<th>Matrix Dimension</th>
<th>Computation Time</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>19.3\text{us}</td>
<td>51\text{us}</td>
</tr>
<tr>
<td>24</td>
<td>44.6\text{us}</td>
<td>78.1\text{us}</td>
</tr>
<tr>
<td>36</td>
<td>111.1\text{us}</td>
<td>153.7\text{us}</td>
</tr>
<tr>
<td>48</td>
<td>220.8\text{us}</td>
<td>265.4\text{us}</td>
</tr>
<tr>
<td>72</td>
<td>611.5\text{us}</td>
<td>675\text{us}</td>
</tr>
<tr>
<td>96</td>
<td>1.3\text{ms}</td>
<td>1.4\text{ms}</td>
</tr>
<tr>
<td>128</td>
<td>2.83\text{ms}</td>
<td>3\text{ms}</td>
</tr>
<tr>
<td>144</td>
<td>3.91\text{ms}</td>
<td>4.07\text{ms}</td>
</tr>
</tbody>
</table>

When compared with some previously published hardware implementation, the performance boost of this RASP based matrix inversion is signifi-
In [9], a FPGA based efficient matrix inversion procedure is described using QR decomposition. The computation time of a 23×23 matrix inversion on Xilinx Spartan3 XC3S1000 is 253us. Another hardware implementation based on the GJ elimination algorithm had been achieved in [16]. The time used by the FPGA to calculate a 36×36 matrix inversion is about 250us. It should be noticed that most of these published hardware implementation is aiming at real matrix, while ours is for complex matrix with single-precision floating-point representation.

In addition, we choose a high performance multicore processor of NXP for comparison. NXP T4240 [17] combines 12 e6500 Power Architecture processor cores with AltiVec and shipped in 2013. The e6500 cores fully resourced dual threads provide 1.7 times the performance of a single thread. The test platform is based on the QorIQ T4240 development system (T4240QDS), and the operating system is VxWorks which is the only real-time operating system capable of exercising the full power of the T4240’s acceleration enhancements. The experiment is running on a single core of T4240, and the execution time of a 72×72 matrix inversion is 2.96ms compared to 675us of our design.

6 Conclusion

In this paper, a VLSI implementation of high-performance parallel matrix inversion based on a reconfigurable application specific processor has been proposed. The implementation process is divided into three steps: LU decomposition with partial pivoting, triangular matrix inversion, and matrix multiplication. Each step reuses the RPEs by time-sharing multiplexing technology which takes good advantage of RASP resource. Our implementation can speed up the loop parts by parallel computing, so as to meet the high real-time capability for radar signal processing. An prototype chip is designed and implemented on 40nm CMOS technology, and the testing results show that our implementation can improve the performance of matrix inversion significantly when compared with some previously published work and commercial DSP. The speed-up ratio can achieve 3 times with the increase of matrix order. For a 144×144 matrix, the execution time is 4.07ms at the frequency of 500MHz.

Acknowledgments

This work was supported by National Nature Science Foundation of China under Grant No. 61176024,61370040,61376075; Research Fund for the Doctoral Program of Higher Education of China under Grant No. 2012009110029; The project on the Integration of Industry, Education and Research of Jiangsu Province BY2015069-05, BY2015069-08; The key Research and Development Program of Jiangsu Province under Grant No: BE2015153; Supported by the Fundamental Research Funds for the Central Universities under Grant No.021014380030; The Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD).