International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:3, 2007

A Reconfigurable Processing Element Implementation for Matrix Inversion Using Cholesky Decomposition

Aki Happonen, Adrian Burian, and Erwin Hemming

***Abstract*—**Fixed-point simulation results are used for the performance measure of inverting matrices using a reconfigurable processing element. Matrices are inverted using the Cholesky decomposition algorithm. The reconfigurable processing element is capable of all required mathematical operations. The fixed-point word length analysis is based on simulations of different condition numbers and different matrix sizes.

***Keywords*—**Cholesky Decomposition, Fixed-point, Matrix inversion, Reconfigurable processing.

I. INTRODUCTION

THE VLSI implementations of digital signal processing algorithms usually require fixed-point arithmetic for the sake of chip area, operation speed, and power consumption. The word-length and scale factor determination is important for harvesting the benefits of fixed-point implementation. Several researches have been conducted for optimizing the used signal word-length, with the purpose of hardware implementation costs minimization and optimization time reduction [1]. At the same time progress of wireless telecommunication technologies and several new emerging applications have imposed flexibility requirements for telecommunication equipment. Future wireless cellular base stations should be able to adopt advanced receiver structures like multiuser detection [2] and higher data rates with new emerging multi-antenna systems [3].

These facts are posing more stringent requirements to design processes and design tools, due to the fact that design lead times will be limited by design and verification time. Also design's non-recurring engineering (NRE) cost roughly doubles when moving from one silicon process to a new one, with the major part of the NRE cost increase coming from higher mask costs. The higher total development cost has raised the desire for reconfigurable architectures [4].

During the past years, reconfigurability has raised a major interest for both research and industry. Most of the commercial coarse grain reconfigurable chips are capable of typical arithmetic and logic operations, and they are having a mesh or array architecture [5]. These reconfigurable chips do not offer optimum solutions for algorithms requiring division and square root operations, e.g. as needed for Cholesky decomposition.

The focus of this paper is fixed-point implementation of matrix inversion using Cholesky decomposition. We made our first considerations over this problem in [16], where we have shown that a fractional word length 16-bits is enough for inverting small and medium size matrices with acceptable errors. In [17] direct and iterative methods for matrix inversion were considered. A coarse grain reconfigurable processing element for future base stations equipment implementation has been proposed in [18]. In the current paper we combine our research work in a common framework.

The paper is organized as follows: some related research results and comments on the need for matrix inversion are presented in Section II. The matrix inversion using Cholesky decomposition is described in Section III. Section IV introduces the used reconfigurable processing element architecture and capabilities. In Section V we describe how the system performances are being measured, and several results are given. We conclude our work in Section VI. This last section shortly presents our future research intentions in this field.

II. RELATED RESEARCH

The need for matrix manipulation appears often in science and engineering. Many well-known algorithms, arising in signal processing, communications, parameter optimization, include the problems of solving linear systems of equations (LSE) or matrix inversion (MI). Because these two problems are related, sometimes one is used to solve the other. We note that this is not an efficient way to solve them. E.g., if we assume that MI is computed using Gaussian elimination with partial pivoting (GEPP), computing the solution as $x=\mathbf{A}^{-1}b$, requires $2n^3$ flops, while by applying GEPP directly to the original LSE system requires only $2n^3/3$. Furthermore, not only that the inversion approach for solving LSE is 3 times slower, but it is also much less stable. Most numerical analysts

Manuscript received November 4, 2004.

A. Happonen is with Nokia Technology Platforms, Elektroniikkatie 3, Oulu, Finland. (e-mail: aki.p.happonen@nokia.com).

A. Burian is on leave from Tampere University of Technology, Finland. He is now with Nokia Research Center, Hermia 5B4, Tampere, Finland. (e-mail: adrian.burian@nokia.com).

E. Hemming is with Nokia Research Center, Meesmannstrasse 103, Bochum, Germany. (e-mail: erwin.hemming@nokia.com).

avoid the usage of matrix inversion [6]. This is because inversion is usually more expensive, and less stable. However, there exist practical situations when the computation of the matrix inverse is needed. This happens for instance, for Wiener and Kalman filtering. Also, all similarity transformations use explicit inverses. Examples are in statistics [7], some eigenvalue-related problems [8], or super-conductivity computation.

There are a variety of methods for matrix inversion, usually classified as direct and iterative. Direct methods are distinguished by the fact that they find the correct solution in a finite number of operations. Iterative methods are characterized by an initial estimate of the solution, and subsequent update of the estimate based on the previous estimate and some error measure. In general, iterative methods do not obtain an exact solution in finite time, but they converge to a solution asymptotically. Iterative algorithms are sometimes preferred, especially in problems of medium/large size, due to smaller storage requirements and efficiency of computational time.

Parallel algorithms for matrix inversion and related problems (LSE, memory multiplication MM, determinant) were initially parallel implementations of traditional serial algorithms [9]. Later, the advances in the area of parallel computation brought about the development of algorithms with the purpose of better exploiting parallelism [10] [12]. In practice, the most used algorithms for solving LSE are Gaussian elimination with pivoting, block Gaussian elimination, and their modifications. These are direct algorithms; in finite number of arithmetic operations (flops) they performed exactly, with no errors. Most of the used direct algorithms for dense matrices fit the following pattern: there are $O(n)$ steps, each requiring $O(n^2)$ work, for a total work estimate of $O(n^3)$.

We must mention that there exist direct algorithms that solve LSE and related problems (MI, MM, determinant) in less than $O(n^3)$ operations. Some progress has been made to develop sequential matrix multiplication algorithms with time complexity $O(n^\tau)$, with $2<\tau<3$. Unfortunately, these algorithms are impractical. The difficulties arise from two different directions. Firstly, they require just too many processors to be realistic. Secondly, these methods suffer from numerical instability (e.g., large overhead constants hidden in the big-O notation). Because of these reasons, these fast algorithms had negligible impact on practical computations. In fact, most of the existing literature on parallel algorithms, considers only the standard sequential algorithm for deriving a parallel one. A supplementary current reason is the fact that the crossover values of n at which these fast algorithms start to beat standard methods are exceedingly high.

To obtain efficient parallelization on distributed memory systems, a more powerful communication mechanism is required. Recently, fiber optical busses have emerged as promising networks [15]. It was shown that for all $1 \le p \le n\tau$ with p being the number of processors on a linear array with a reconfigurable pipelined bus system we need $O(n\tau/p + n^2/p^{2/\tau} \log p)$ time. For $O(n\tau)$ processors we need $O(\log n)$ time.

The parallel variants of the sequential Gaussian elimination algorithm for matrix inversion are using $O(\log 2n)$ parallel time and a very large but polynomial in n processor bound [10] [12]. On a distributed memory machine the parallel version of Gaussian elimination has the parallel run time and the required number of processors of $O(n)$. On a systolic array with $n(n+1)/2$ cells, the execution time is $3n$, while on a ring with $n/2$ processors is $2n\varepsilon$ (ε is the required time for the processors in the ring to communicate with each other) [11].

III. MATRIX INVERSION USING CHOLESKY DECOMPOSITION

Given a factorization $PA = LU$, two ways to evaluate A^{-1} exist: as $A^{-1}=U^{-1}\times L^{-1}\times P$, and as the solution of $UA^{-1}=L^{-1}\times P$. These methods generally achieve different levels of efficiency on high-performance computers, and they propagate the rounding errors in different ways. The quality of an approximation $Y \approx A^{-1}$ can be assessed by looking at the right and left residuals, $AY - I$ and $YA - I$, and the forward error, $Y - A^{-1}$.

In terms of the error bounds, there is little to choose among different methods for inverting matrices [7]. Therefore, the choice of the method can be based on other criteria, such as performance and the use of working storage. This is why Cholesky decomposition has been used to implement the matrix inversion.

Any nonsingular matrix $A(n\times n)$ that can be factored in the form R^TR is positive definite. The converse is also true. If A is positive definite, then A can be factored in the form $A = L^TL$, where L is lower triangular. If, in addition, we require the diagonal elements of L to be positive, the decomposition is unique and is called the *Cholesky decomposition* or the *Cholesky factorization* of A. The used algorithm, which uses the same location of A for the result L, is given in Figure 1.

```
for j = 1:n,
  if j > 1;
    A(j:n, j)= A(j:n, j) - A(j:n, 1:j-1)*A(j, 1:j-1)';
  end;
  A(j:n, j)=A(j:n, j)/sqrt(A(j, j));
end;
```

Figure 1. Cholesky-factorization – Matlab Example.

Given the Cholesky decomposition of A, we solve the linear system $Ax = b$ by solving the two triangular systems

$$1.\ L^Ty = b \quad \text{and} \quad 2.\ Lx=y.$$

A triangular system requires $\frac{1}{2}n^2$ operations to solve, and the two systems together require n^2 operations. To the extent that the operation counts reflect actual performance, we will spend more time in the Cholesky algorithm when $1/6n^3 > n^2$, or when $n > 6$. For somewhat larger n, the time spent solving the triangular systems is insignificant compared to the time spent computing the Cholesky decomposition. In particular,

having computed the Cholesky decomposition of a matrix of moderate size, we can solve several systems having the same matrix at practically no extra cost. At the beginning of the Section II we have deprecated the practice of computing a matrix inverse to solve a linear system. Now we can see why.

After the decomposition is done, computing the inverse of a triangular matrix and multiplying it with its transpose can do computing the inverse. The used algorithm computes the columns of $X=L^{-1}$ in reverse order and it is shown in Figure 2.

```
for j = n:-1:1,
   X(j,j) = 1/L(j, j);
   for k = j+1:n
      for i = j+1:n
         X(k, j) = X(k, j) + X(k, i)*L(i, j);
      end;
   end;
   for k = j+1:n
      X(k, j) = -X(j, j)*X(k, j);
   end;
end;
```

Figure 2.Inverse of a triangular matrix – Matlab example.

IV. Architecture Overview

In this section, we describe the used matrix inversion engine. This engine is implemented using a reconfigurable processing element plotted in

Figure 3. The complexity analysis for a 12 bits implementation can be found in [18].

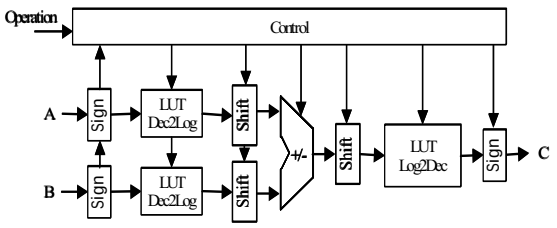

Figure 3. Processing Element Architecture.

The architecture has fixed latency for following mathematical operations: addition, subtraction, multiplication, division and square root. In addition to these operations, conjugated equations, e.g. $C= A/(B^{1/2})$, can be executed every clock cycle with 3 clock cycles latency.

TABLE I shows the supported arithmetic operations, their implementation, and the operation command having A and B as inputs and C as the output of the processing element. By using the logarithmic arithmetic for multiply and division, the implementation uses only adders. Similarly implementations for square and square root are derived using shifters.

TABLE I.
Processing Element Equations

Function	Mathematical equation in used architecture	Operation Command
Square root	$C=2^{(\log_2(A)/2)}$	000
Square	$C=2^{(2*\log_2(A))}$	001
Multiplication	$C=2^{(\log_2(A)+\log_2(B))}$	010
Division	$C=2^{(\log_2(A)-\log_2(B))}$	011
Add	C=A+B	100
Subtract	C=A-B	101
Conjugated square root and division	$C=2^{(\log2(B)-\log_2(A)/2)}$	110

A. Fixed point implementation

We created a simulation environment for the analysis of word length effects over matrix inversion performances. The base architecture of the reconfigurable processing element is the same for all word lengths. The look-up tables (LUT) contain two read only memories (ROM), scaled according to the required accuracy. We have used the following ROM sizes: 64x17 bits for Dec2Log, and 64x14 bits and Log2Dec. The output of the reconfigurable processing element is truncated to the reported number of bits.

For inverting a triangular matrix we have changed the position of the zeros in order to increase the number of effective bits. This scaling does not affect the used word length. We have assumed that the proposed reconfigurable processing architecture is a hardware accelerator (HWA) for the main processor and main processor is able to scale down result to right level. All the needed scaling has been implemented using shifters.

V. Simulation Results

For reporting the results the residual $Z = A*\text{inv}(A)-I$ has been used (since our input matrices are positive definite, the other residual is not needed). The computation has been considered successful if the 2-norm of the residual is less than a predefined error level. The used error levels have been $\varepsilon_k = 2^{-k}$, k = 0,1,…,5. Also the forward error has been computed using the Matlab environment in order to show the differences between fixed-point and floating-point solutions. The maximum and mean values of forward errors are given.

The results from TABLE II represent the percentage of successful inverse computations for 100 input matrices, versus the specified error level. We conclude from these results that the performance of matrix inversion engine decreases with the condition number of the input matrices, as we expected. The forward error values for condition numbers between 200-300 are shown in Figure 4, for the same test case presented in TABLE II. We note that even if the forward error is relatively small, the residual errors are significant.

We have also analyzed the effect of word length for matrix computing engine performance. TABLE III shows the results of different word lengths for 16x16 matrices having condition number less than 200. The simulation results are percentages of successful inverse computations for 100 randomly

generated input matrices versus the specified error level.

VI. CONCLUSION AND FUTURE RESEARCH

In this paper, a fixed-point implementation of matrix inversion using reconfigurable HW processing element to perform Cholesky decomposition has been presented. Experiments using several fixed-point word lengths have been realized. By using our reconfigurable processing element with a 16-bits word length, the obtained errors are acceptable for small size matrices with low condition numbers. Higher word length is needed to invert larger matrices or higher condition numbers.

In the future, we are planning to use the proposed reconfigurable processing element in selected wireless applications. One target for our future research is to study the matrix inversion performances as part of a wireless receiver, and to establish the word length requirements based on the characteristics of matrices to be inverted in this receiver.

TABLE II
16 BITS IMPLEMENTATION RESIDUALS FOR 8X8 MATRICES WITH DIFFERENT CONDITION NUMBERS.

	Condition Numbers			
Error	0-50	50-100	100-200	200-300
ε_0	100	100	85	40
ε_1	100	82	60	12
ε_2	82	33	9	0
ε_3	14	0	0	0
ε_4	0	0	0	0

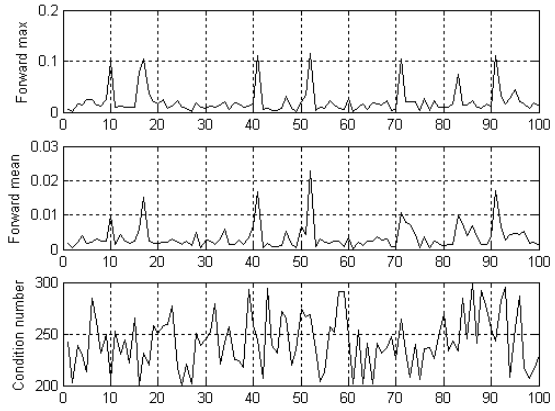

Figure 4. Forward Error and Condition Numbers.

TABLE III
RESIDUALS FOR 16X16 MATRICES WITH DIFFERENT WORD LENGTHS.

Error	16 bits	20 bits	24 bits
ε_0	79	99	100
ε_1	65	85	97
ε_2	28	22	80
ε_3	1	0	25
ε_4	0	0	1
ε_5	0	0	0

REFERENCES

[1] Ki-Il Kum and Wonyong Sung, **"**Combined word-length optimization and high-level synthesis of digital signal processing systems**",** *IEEE Tran. On Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 8, Aug. 2001, pp. 921-930.

[2] M. J. Juntti, **"**Performance analysis of linear multisensor multiuser receivers for CDMA in fading channels"**,** *IEEE Journal on Selected Areas in Communications*, vol. 18 , no. 7 , July 2000, pp. 1221 – 1229.

[3] G. J. Foschini and M. J. Gans, "On limits of wireless communications in a fading environment when using multiple antennas", *Wireless Personal Communications*, vol. 6, no.3, 1998

[4] R. Baines and D. Pulley, "A Total Cost Approach to Evaluating Different Reconfigurable Architectures for Baseband Processing in Wireless Receivers", *IEEE Communication Magazine*, January 20003, Page(s): 105-113

[5] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective", Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings, 13-16 March 2001, Page(s): 642 –649

[6] N.J. Higham, Accuracy and Stability of Numerical Algorithms, SIAM, Philadelphia, 1996.

[7] P. McCullagh and J.A. Nelder, *Generalized Linear Models*, Chapmann and Hall, London, 1989.

[8] R. Byers, "Solving the algebraic Riccati equation with matrix sign function"*, Linear Algebra Applications*, vol. 85, 1987, pp. 267-279.

[9] G.W. Stewart, *Afternotes on Numerical Analysis*, SIAM, Philadelphia, 1996.

[10] M. Cosnard and D. Trystram, *Parallel Algorithms and Architectures*, Blackwell North America, Inc., 1995.

[11] A. El-Amawy, "A Systolic Architecture for Fast Dense Matrix Inversion", *IEEE Trans. Computers*, 38, no. 3, pp. 449-455, 1989.

[12] V. Kumar, A. Grama, A. Gupta, and G. Karypis*, Introduction to Parallel Computing. Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company, Inc., 1994.

[13] E. Dekel, D. Nassimi, and S. Sahni, "Parallel Matrix and Graph Algorithms", SIAM J. Computing, vol. 10, pp. 657-673, 1981.

[14] H. Park, H.J. Kim, and V.K. Prasanna, "An O(1) time optimal algorithm for multiplying matrices on reconfigurable mesh", Information Processing Letters, vol. 47, pp. 109-113, 1993.

[15] K. Li and Y. Pan, "Parallel Matrix Multiplication on a Linear Array with a Reconfigurable Pipelined Bus System", IEEE Trans. Computing, vol. 50, no. 5, pp. 519-525, 2001.

[16] A. Burian, J. Takala, and M. Ylinen, "A fixed-point implementation of matrix inversion using Cholesky decomposition", The 46th International Midwest Symposium On Circuits and Systems, MWSCAS, December 27-30, 2003, Cairo, Egypt.

[17] M. Ylinen, A. Burian, and J. Takala, "Direct versus iterative methods for fixed-point implementation of matrix inversion", Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on , Volume: 3 , 23-26 May 2004, Pages:III - 225-8 Vol.3.

[18] A. Happonen, E. Hemming, and M.J. Juntti, "A novel coarse grain reconfigurable processing element architecture", The 46th International Midwest Symposium On Circuits and Systems, MWSCAS, December 27-30, 2003, Cairo, Egypt.