International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:1, No:11, 2007

A Reconfigurable Processing Element for Cholesky Decomposition and Matrix Inversion

Aki Happonen, Adrian Burian, and Erwin Hemming

***Abstract*—**Fixed-point simulation results are used for the performance measure of inverting matrices by Cholesky decomposition. The fixed-point Cholesky decomposition algorithm is implemented using a fixed-point reconfigurable processing element. The reconfigurable processing element provides all mathematical operations required by Cholesky decomposition. The fixed-point word length analysis is based on simulations using different condition numbers and different matrix sizes. Simulation results show that 16 bits word length gives sufficient performance for small matrices with low condition number. Larger matrices and higher condition numbers require more dynamic range for a fixed-point implementation.

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

I. INTRODUCTION

THE VLSI implementations of digital signal processing algorithms usually require the usage of 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 the 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, the progress of wireless telecommunication technologies and several new emerging applications have imposed flexibility requirements to telecommunication equipments. Future wireless cellular base stations should be able to use 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 integrated circuits 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 was 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 and we consider the performance of reconfigurable processing engine for matrix inversion.

The paper is organized as follows: some related research results and comments on the need for matrix inversion are given in Section II. The matrix inversion using Cholesky decomposition and the matrix condition number are described in Section III. Section IV introduces the used reconfigurable processing element architecture and its 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

Manuscript received November 5, 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).

slower, but it is also much less stable. Most numerical analysts 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. Other examples are in statistics [7], some eigenvalue-related problems [8], or super-conductivity computation [19].

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 a 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, matrix multiplication, 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 of size $n \times n$ 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, matrix multiplication, 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., given by 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.

On distributed memory processors it was shown that we need $O(n^3/p+\log(p/n^2))$ time on a hypercube with p processors, where $n^2 \le p \le n^3$ [13]. Matrix multiplication can be done in constant time on a reconfigurable mesh with n^4 processors [14]. But, such implementations, though very fast, are far from cost-optimal for medium and large size matrices.

To obtain efficient parallelism 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, $Y - A^{-1}$.

Suppose we perturb $A \to A + \delta A$ with $|\delta A| \le \varepsilon |A|$; thus, we are making relative perturbations of size at most ε to the elements of A. If $Y = (A+ \delta A)^{-1}$ then $(A + \delta A)Y = Y(A + \delta A) = I$, so that

$$\left|AY - I\right| = \left|\delta A \cdot Y\right| \le \varepsilon \left|A\right|\left|Y\right| \quad (1)$$

$$\left|YA - I\right| = \left|Y \cdot \delta A\right| \le \varepsilon \left|Y\right|\left|A\right| \quad (2)$$

and, since $(A + \delta A)^{-1} = A^{-1} - A^{-1}\delta A\, A^{-1} + O(\varepsilon^2)$,

$$\left|A^{-1} - Y\right| \le \varepsilon \left|A^{-1}\right|\left|A\right|\left|A^{-1}\right| + O(\varepsilon^2). \quad (3)$$

These bounds represent "ideal" bounds for a computed approximation Y to A^{-1}, if we regard ε as a small multiple of the unit round off u. For triangular matrix inversion, appropriate methods do indeed achieve (1) or (2) but not both, and (3). It is important to note that neither (1), (2), nor (3) implies that $Y + \delta Y = (A + \delta A)^{-1}$ with and that is, Y need not be close to the inverse of a matrix near to A, even in the norm sense. Indeed, such a result would imply that both the left and right residuals are bounded in norm by $(2\varepsilon+\varepsilon^2)\|A\|_\infty \|Y\|_\infty$ and this is not the case [7].

In terms of the error bounds, there is little to choose among different methods for inverting matrices. Therefore, the choice of the method can be based on other criteria, such as performance and the use of working storage. These are two of the reasons why we have chosen the Cholesky decomposition to implement the matrix inversion.

A. Cholesky Decomposition

Any nonsingular matrix $A(n\times n)$ that can be factored in the form R^TR is positive definite. The converse is also true. We note that in order to use the Cholesky decomposition the input matrix should be positive definite. Otherwise, two supplementary matrix multiplications are needed (i.e., if B is a general dense nonsingular matrix then $A = B^TB$ is positive definite, and $B^{-1} = (B^T B)^{-1} B^T = A^{-1}B^T$).

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;
```

Fig. 1 Cholesky-factorization – Matlab Example

The algorithm proceeds in n stages. At the first stage, the first row of R is computed and the $(n - 1)\times(n - 1)$ matrix A* in the southeast corner is modified. At the second stage, the second row of R is computed and the $(n - 2)\times(n - 2)$ matrix in the southeast corner is modified. The process continues until it falls out of the southeast corner. Thus the algorithm begins with a loop on the row of R to be computed.

A second possibility is to work with the upper half of the array A and compute $R = L^T$. For then the rows of L become columns of R.

If A is ill-conditioned, the computed factor $\widetilde{L}$ will not generally be close to the true factor L; the best we can say is that $\left|\widetilde{L} - L\right| \Big/ \left|L\right| = O\left(\kappa(A)\varepsilon_m\right)$, where $\kappa(A)$ is the condition number of matrix A, and ε_m is the machine epsilon. In other words, Cholesky factorization is in general an ill-conditioned problem. It is only the product $\widetilde{L}^*\widetilde{L} = A + \delta A$, which satisfies the much better error bound $\left|\delta A\right| \Big/ \left|A\right| = O(\varepsilon_m)$. Thus, the errors introduced in $\widetilde{L}$ by rounding are large but ``diabolically correlated" [9]. This means that if testing after the Cholesky decomposition the ``true error" is not obtained. Only after the inverse is computed, the effect of these errors can be noticed at their true values. In other words, for a correct evaluation of the performances we need to use the forward error and the residuals.

Given the Cholesky decomposition of A, we solve the linear system $Ax = b$ by solving the two triangular systems: $L^Ty = b$ and $Lx=y$.

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.

A good way to calculate the inverse $X = (x_1\ x_2\ \dots\ x_n)$ of a symmetric positive-definite matrix A is to compute the Cholesky decomposition and use it to solve the systems

$$Ax_j = I_j, \qquad j = 1,2,...,n,$$

where I_j is the jth column of the identity matrix I. Now if these solutions are computed in the most efficient way, they require $1/3n^3$ additions and multiplications - twice as many as the Cholesky decomposition. Thus the invert-and-multiply approach is much more expensive than using the decomposition directly to solve the linear system [10].

B. Inverting a Triangular Matrix

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;
```

Fig. 2 Inverse of a triangular matrix – Matlab example

C. The Condition Number

In most of the numerical analysis literature, the complexity and stability of numerical algorithms are usually estimated in terms of the problem instance dimension and of a 'condition number'. The complexity of solving an $n\times n$ linear system **Ax=b** is usually estimated in terms of the dimension n (actually the input size is $n(n+1)$) and of the condition number. The condition number (with respect to the 2-norm) of a matrix **A** with respect to inversion is defined as

$$\kappa(A) = \|A\| \cdot \left\|A^{-1}\right\|.$$

Although it looks like you need to compute a matrix inverse to compute the condition number, there are reliable ways of estimating it from the LU decomposition [9].

There exist a set of problems instances with $\kappa(A)=\infty$ and in most cases it makes no sense to attempt solving them. There are also problem instances (in our case, matrices) close to the locus of the degenerate problem instances. Those will have a large condition number, and will be said to be *ill-conditioned*. It is generally accepted that ill-conditioned problem instances are hard to solve. Thus, for complexity purposes, a problem instance with a large condition number should be considered 'large'. Therefore, when considering problems defined for real inputs, a reasonable measure for the input size would be $n^2\log_2\kappa(A)$.

Another tradition, derived from classical complexity theory and pervasive in several branches of literature (such as linear programming), is to consider the subset of problems instances with integer coefficients. Hence the input size is the number of coefficients times the bit-size of the largest coefficient in absolute value.

The condition number is always greater than one:

$$1 \le \|I\| \le \left\|A \cdot A^{-1}\right\| \le \|A\| \cdot \left\|A^{-1}\right\| = \kappa(A).$$

This means that the condition number is a magnification constant: the bound of the error is never diminished in passing from the matrix to the solution.

Suppose that **A** is rounded on a machine with rounding unit ε_M, so that $\tilde{a}_{ij} = a_{ij}(1 + \varepsilon_{ij})$, where $\varepsilon_{ij} \le \varepsilon_M$. If we solve the linear system $\tilde{A}\tilde{x}=b$ without further errors, we get a solution that satisfies

$$\frac{\left\|\tilde{x} - x\right\|}{\|x\|} \le \kappa(A)\varepsilon_M .$$

In particular, if $\varepsilon_M=10^{-t}$ and $\kappa(A)= 10^k$, the solution $\tilde{x}$ can have relative error as large as 10^{-t+k}. Thus, the larger components of the solution can be inaccurate in their $(t-k)$th significant figures. Unfortunately, the relative error gives a good idea of the accuracy of the larger components but says little about small components. So, the smaller components can be much less accurate. This justifies the following rule:

If $\kappa(A)= 10^k$ expect to lose at least k digits in solving the system $\mathbf{Ax} = \mathbf{b}$. We note that this discussion was done in the context of solving linear systems just because it is usually done in this context in the literature. Because of the connection with the matrix inverse, these facts are valid in this case too, even if the matrix inverse is determined out of the context of solving linear systems. We also note that a lot of attention should be paid to properly scaling the given **A** matrix. When you get a large condition number, you have to go back to the original problem and take a hard look to see if it is truly sensitive to perturbations or is just badly scaled [9].

IV. Architecture Overview

In this section, we briefly introduce the used matrix inversion engine. The matrix inversion engine has been implemented using a reconfigurable processing element illustrated in Figure 3. A detailed complexity analysis for 12 bits implementation is given in [18].

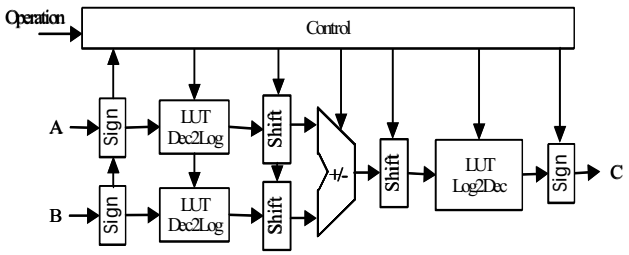

Fig. 3 Processing Element Architecture

The most complex blocks of the reconfigurable processing element are the look-up-tables (LUTs), which convert the element's input and output data from decimal scale to logarithm scale and vice versa. Their implementation is based on compressed read-only memories (ROM) and combination logic. The architecture of a LUT to compute the base 2 logarithm of its input, Dec2Log, is given in Figure 4.

The LUTs contain two ROMs, one for coarse value and the other for fine value of output, implemented as a combinatorial logic. We have utilized the nature of base of 2 logarithm in implementing the ROM compression. The base of 2 logarithms from decimal number increases by one when decimal number doubles, i.e., the logarithm increases by one between any consecutive powers of two. By knowing this, we can compress the LUTs having one-to-one value for input data from 2^{N-2} to 2^{N-1}-1, where N is the data width of the input. The rest of the input values have been constructed by scaling. Figure 5 shows four examples of logarithm behavior between two consecutive powers of two, each having one hundred intermediate steps. By utilizing this feature of the logarithm, we can implement any word length without the need of changing the underlying architecture. The LUT that performs 2 to power of number (Log2Dec) has a similar architecture.

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

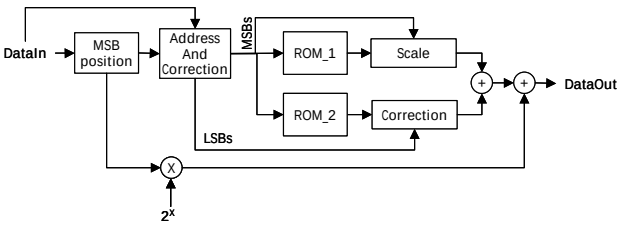

Fig. 4 LUT Architecture

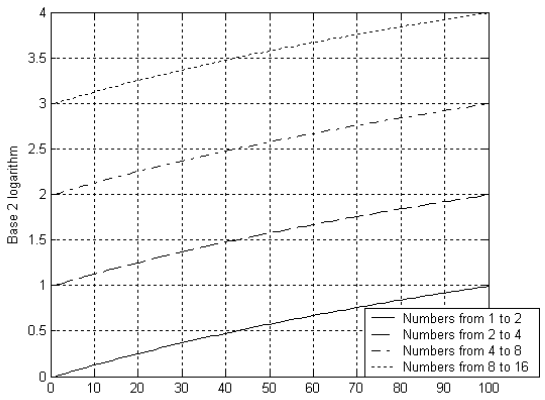

Fig. 5 Logarithm values between two consecutive powers of two

TABLE I shows the supported arithmetic operations of the processing element and their implementation having two data inputs, A and B, and one data output, C. We note that a major advantage of the used architecture is the implementation of the logarithmic arithmetic for multiplication and division through additions and subtractions, and similarly implementation for square and square root through shifting operations.

TABLE I PROCESSING ELEMENT EQUATIONS

Function	Mathematical equation in used architecture	Control 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	$C=2^{(\log_2(B)-\log_2(A)/2)}$	110

The processing element contains the needed set of arithmetic operations for implementing the Cholesky decomposition algorithm and for computing the inverse of a triangular matrix. Furthermore, the conjugated square root and one division operation, both required by the Cholesky decomposition algorithm, can be performed in single clock cycle.

V. SIMULATION RESULTS

We created a simulation environment for realizing the analysis of the effect of the used word length over the performance of the matrix inversion. The base architecture of reconfigurable processing element is the same for all word lengths. For implementing the LUTs, we have used ROM sizes of 64x17 bits for Dec2Log and 64x14 bits for Log2Dec. The inputs and the output of the reconfigurable processing element are truncated to the reported number of bits.

For implementing the triangular matrix inversion, we have changed the position of the zero to have a larger number of effective bits. This scaling does not affect the used word lengths. We have assumed that the proposed architecture is a hardware (HW) accelerator for the main processor, and that the main processor is able to scale down the obtained results for further processing. The scaling has been implemented by shifting.

For reporting the results, only the right residual $Z=|A*Y-I|$ has been used, because since our input matrices are positive definite, the left 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. We have also used the forward error, $|Y - A^{-1}|$, in order to illustrate the differences between fixed-point and floating-point solutions. The maximum and the mean values of the forward error 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 the 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 6, while the same test case residuals are presented in TABLE II. We note that in some cases, even if the forward error is relatively small, the residual errors are significant.

We have also analyzed the effect of word length over the matrix computing engine performance. TABLE III shows the effect of using different word lengths for the inversion of 16x16 matrices having condition numbers limited to be less than 200. The simulation results are percentages of successful inverse computations for 100 randomly generated input matrices versus the specified error level.

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

Error	Condition Numbers			
	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

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

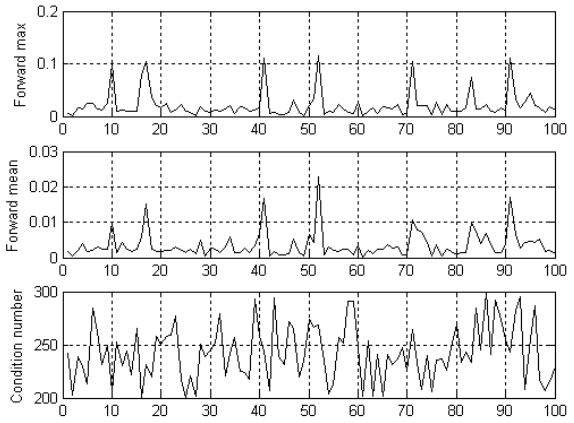

Fig. 6 Forward Error and Condition Numbers

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. Wider word lengths are 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 and throughput requirements based on the characteristics of matrices to be inverted in this receiver.

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 2003, 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 Circuits and Systems, Volume: 3 , 23-26 May 2004, pp. 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.

[19] W. Benzing, T. Scherer, W. Jutzi, "Inversion calculation of two dimensional current distributions from their magnetic field*",IEEE Transactions on Applied Superconductivity,* vol. 3, issue 1, March 1993, pp. 1902-1905.

Aki Happonen was born in Finland on August 6, 1969. He received the M.Sc. degree in information technology in 1994 from Tampere University of Technology, Finland. At the moment he is Senior Technology Manager in Nokia Technology Platforms. His areas of interest are related to wireless receiver implementation issues.

Adrian Burian was born in Romania on April 24, 1969. He received the M.Sc. degree in applied electronics in 1993, the Ph.D. degree in Electronics and Telecommunications in 2000 from Technical University of Cluj-Napoca, Romania, and the Dr.Tech. degree in Information Technology in 2002 from Tampere University of Technology, Finland.
Between January and April 1996 and during March and April 1997, he was a visiting researcher at the Microelectronics Laboratory, Catholic University of Louvain, Louvain-la-Neuve, Belgium, and the University of Zaragoza, Spain, respectively. From 1994 to 1997, he was a Teaching Assistant with the Department of Bases of Electronics at the Technical University of Cluj-Napoca. Since 1998, he has been a Researcher at the Tampere University of Technology, Tampere, Finland, where he is currently on leave. From April 2004 he has been with Nokia Research Center, Tampere, Finland. His research interests include nonlinear signal and image processing, parallel processing, computer systems, and neural networks.

Erwin Hemming was born in Germany on September 7, 1963. He received his Dipl.-Ing. degree in Electrical Engineering in 1990 and Ph.D. degree in 1998 from the University of Duisburg, Germany. From 1990 to 1996, he worked at the Fraunhofer Institute of Microelectronic Circuits and Systems, Duisburg. He was engaged in research of a Smart-Power circuit, SOI analog and digital components, single chip inkjet-head and a digital ATM-chip. He joined Nokia Research Center in 1996. His current research activities focus on radio physical layer architectures and physical implementation of WCDMA/HSDPA and Global Positioning Systems (GPS). For the last five years he has been working in the area of developing digital architecture and components for ASICs used in wireless radios, GSM/Edge and 3G Terminal and base station and WLAN IEEE802.11a.