2015 International Conference on Computational Science and Computational Intelligence

Implementation Aspects of the Matrix Inverse Computation and Inverse Computations Completness Method

Etienne Aubin Mbe Mbock *
Computer Engineering and Computer Sciences
ACM Member Student(Computing, 7127189)
Eichendorffstrasse 35, 78120 Furtwangen
Im Schwarzwald, Deutschland
E.Mbe_Mbock@stud.uni-heidelberg.de
Mbemb_ock@yahoo.com

Abstract— It becomes interesting to analyze and apply the features of reconfigurable computations because the concept allows algorithm creation. An important algorithm that resulted from this concept is the known matrix inverse computation. This algorithm is becoming common for matrix based computations because it is free from singularities in comparison with other "Gauß" methods. However, this extreme ease computational requirement has a limitation. The generated matrix inverse are all upper triangular or lower triangular. Since there is a need to extend computations to any matrix, we then present some implementation aspects of the reconfigurable matrix inverse and an extension of the process that handles full matrix inverse. This research uses the results of the reconfigurable matrix inverse computations completes them and makes the process capable of generating full matrix inverse.

Keywords—Matrix Inverse Computation, Recursive Linear Process, Reconfiguration, Algorithms, Code Generation.

INTENSIVE research on reconfiguration systems and computations have been deployed in the past years. Many of these techniques concentrate on hardware [1]–[3] and applied in On Chip-Networks and operating systems [4], [5]. Most recently significant research has been achieved in algorithms and computations reconfiguration [6]–[9]. In most of these applications it is not described effectively how their process are constructed, this is due to the fact that, they emphasize on their specific applications. This paper will consider the dynamic equations of the linear recursive process summarized by the following extended state equation.

$$\begin{bmatrix} q_2 \\ q_3 \\ \vdots \\ \vdots \\ q_N \end{bmatrix} = q_1 \begin{bmatrix} \alpha_{2,1} \\ \alpha_{3,1} \\ \vdots \\ \vdots \\ \alpha_{N,1} \end{bmatrix} + q_2 \begin{bmatrix} 0 \\ \alpha_{3,2} \\ \alpha_{4,2} \\ \vdots \\ \alpha_{N,2} \end{bmatrix} + \cdots + q_{N-1} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \alpha_{N-1,N-1} \\ \alpha_{N,N-1} \end{bmatrix} + q_N \begin{bmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ \alpha_{N,N} \end{bmatrix} \tag{1}$$

We admit the given process and the proposed algorithm without demonstrations for more details see http://world-comp.org/p2013/PDP.html. We also admit that all matrix and vector operations are all well defined and the size of all provided matrices and vectors in all cases is suitable for computations. We also admit that any $n \times n$ matrix

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n-1} & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n-1} & a_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn-1} & a_{nn} \end{bmatrix}$$

can be reconfigured in two inverse matrices R and V of the same size. For more details on the mathematical background please consult [9]. We will admit furthermore without demonstration that any matrix inverse can be transposed. The transpose of a row vector A_j is the column vector A_j^t and the transposed of a matrix A is denoted A^t.

1 Instructions for Implementation

1.1 Programming Features

The reconfiguration of the Recursive Linear Process gives the matrix inverse computation. In Programming this inverse matrix computations parallel features have been considered. To guarantee the efficiency, all program fragments have been first checked by hand and our programming approach subscribes to the following universal criteria:

1) Comfortably and easy and ready to use

2) Standard and permits the re-entrance of data and can be easily used by language programmers

3) Easily translate-able to any programming language.

The listing that follows represent the code that we generated by using the "generated by Norcroft ARM C". Describing this code reveals the following architectural information: we count number of registers in the implementation to be fifteen see Table 1. These are divided into three groups. The first group contains arguments variables specified in the code by letter a. The second group is made of register variables labelled v. The third group is divided into static base register variables named sb, v6 as stack limit register variable, the frame pointer register variable is fp the stack pointer frame sp, the link address register lr and the program counter labelled pc. The intructions are divided from 1 to 5 arguments instruction. The instruction LDMDB for instance takes the following arguments fp,v1-v6,fp,sp,pc. The values of these registers are summarized in the coming table and for this application, the value of the program counter is set to pc=0x00008ce0. The last part of the implementation is reserved to the data entries. For demonstrative purpose, we assume that these entries are limited to 25 real numbers summarized into A and B.

978-1-4673-9795-7/15 $31.00 © 2015 European Union
DOI 10.1109/CSCI.2015.31

r0	0xffffffff
r1	0x00000000
r2	0x00000000
r3	0x00000000
r4	0x00000000
r5	0xffffffff
r6	0x00000f0f
r7	0x00000000
r8	0x00000001
r9	0x0000000e
r10	0x7ffff230
r11	0x00000000
r12	0x00000000
r13	0x80000000
r14	0x00008d18

TABLE 1: Values of the Fifteen Registers.

The values of the codes around some addresses can be viewed, in Figure 1. Figure 1 presents the assembly code at the following chosen addresses: 0x00000000, 0x00008008, 0x0000a44.

Listing 1: Implementation

```
main
        MOV      ip,sp
        STMDB    sp!,{v1-v6,fp,ip,lr,pc}
        SUB      fp,ip,#4
        CMP      sp,sl
        BLMI     __rt_stkovf_split_small
        SUB      sp,sp,#8
        MOV      v1,#5
        MOV      v2,#0
|L000020.J4.main|
        MOV      a1,#0
        CMP      v1,#0
        BLE      |L00005c.J8.main|
        LDR      lr,[pc, #L0002cc-.-8]
        LDR      v6,[pc, #L0002d0-.-8]
|L000034.J7.main|
        ADD      a2,a1,a1,LSL #2
        ADD      a3,v6,a2,LSL #3
        ADD      a3,a3,v2,LSL #3
        ADD      a2,lr,a2,LSL #3
        ADD      a2,a2,v2,LSL #3
        LDMIA    a2,{a4,ip}
        STMIA    a3,{a4,ip}
        ADD      a1,a1,#1
        CMP      a1,v1
        BLT      |L000034.J7.main|
|L00005c.J8.main|
        MOV      v3,#0
        SUBS     a1,v2,#1
        STR      a1,[sp,#4]
        BMI      |L000144.J12.main|
|L00006c.J11.main|
        ADD      a1,pc,#L0002d4-.-8
        MOV      v6,#0
        CMP      v1,#0
        LDR      v4,[a1,#4]
        LDR      v5,[a1,#0]
        BLE      |L0000cc.J15.main|
|L000084.J14.main|
        ADD      a1,v6,v6,LSL #2
        LDR      a2,[pc, #L0002dc-.-8]
        ADD      a2,a2,a1,LSL #3
        ADD      a4,a2,v2,LSL #3
        LDR      a2,[pc, #L0002d0-.-8]
        ADD      a1,a2,a1,LSL #3
        ADD      a2,a1,v3,LSL #3
        LDMIA    a2,{a1,a2}
        LDMIA    a4,{a3,a4}
        BL       _dmul
        MOV      a3,v5
        MOV      a4,v4
        BL       _dadd
        MOV      v5,a1
        MOV      v4,a2
        ADD      v6,v6,#1
        CMP      v6,v1
        BLT      |L000084.J14.main|
```

We used the ARM Toolkit v2.02, to implement the Matrix Inverse Method and generated the standalone code. Listing 1 is a peace of the generated code. We used the algorithm proposed in [9] to ensure that the code generated is correct. We have generated a listing as an assembly code of the process. The complete listing initializes the matrices A and B as input arguments. The matrices that have been input as test are organized in a pascal matrix of order 5 and identity matrix of order 5. Arrays of size 5×5 are reserved to the variables R, V. They will contain the matrix entries of the matrix inverse computations. We created variables R1 and R2 that are initialized with zeros. The main part of the program in the complete listing is made of a for loop that iterates on the index variable k. There are 8 other for loops that will compute the following data:

1) The variable V[s][k] is assigned the value of B[s][k] in the first subloop
2) In the second subloop, we iterate through the index variable j incrementing it by one each time and stop if the index variable reaches n-1. If the variable is increased up to n then this will result in unexpected computations and the objectives matrices will not be reached. This for loop contains two other subloops:
   a) The first subloop is design to fill inside the variable V[s][k] values of B[s][k]
   b) The second subloop we want to reach stage n-1 that is we iterate through the index j
   c) The third subloop will provide the computations of the diagonal of the matrix R
   d) The fourth and last for subloop computes the matrix V
3) The sixth for loop is a print statement that will output the matrix R
4) The seventh and eight last for loops prints out the matrix V and complete the Assembly description of the Listing.

The generated listing can also be used for the Recursive Linear Process. The number of loops and subloops will be the same.

2 Completness Method

The previous sections points at partial reconfiguration and matrix inverse computation. The state-of-the-art implementation is performed using [9]. This makes use of the recursive

$1.00000e+00$	$1.00000e+00$	$1.00000e+00$	$1.00000e+00$	$1.00000e+00$
$0.00000e+00$	$1.41421e+00$	$1.41421e+00$	$2.12132e+00$	$2.82843e+00$
$0.00000e+00$	$0.00000e+00$	$1.41421e+00$	$4.24264e+00$	$7.07107e+00$
$0.00000e+00$	$0.00000e+00$	$0.00000e+00$	$3.53553e+00$	$-5.65685e-01$
$0.00000e+00$	$0.00000e+00$	$0.00000e+00$	$0.00000e+00$	$6.53605e+00$

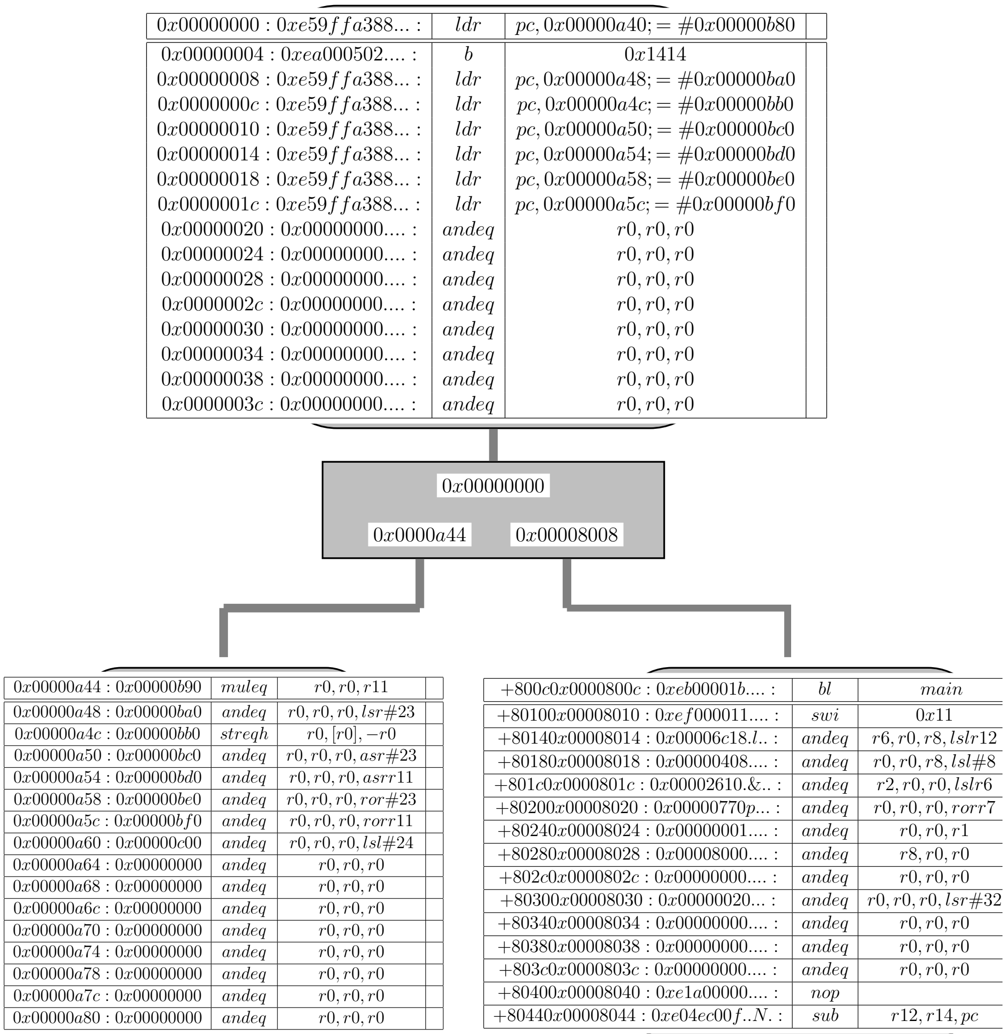

Fig. 1: Reconfigurable Matrix Inverse. Upper side, the Code around location 0x00000000. The right down side identifies the code around location 0x00008008. The left down side is the code around location 0x0000a44.

linear process stated in equation 1. In this section, we propose the general matrix inverse computation. We consider discrete matrices given by:

1) $B_1 = \begin{pmatrix} B_{1\,j} & B_{1\,1} \end{pmatrix} \qquad j = 1, 2, 3, \cdots, n$

$$
2)\quad B_2 = \begin{pmatrix} B_{1\,1}B_{1\,2} \\ \\ B_{1\,2}B_{1\,2} + B_{2\,2}B_{2\,2} \\ \\ B_{1\,j}B_{1\,2} + B_{2\,j}B_{2\,2} \end{pmatrix} \qquad j = 3, 4, \cdots, n
$$

$$
3)\quad B_3 = \begin{pmatrix} B_{1\,1}B_{1\,3} \\ \\ B_{1\,2}B_{1\,3} + B_{2\,2}B_{2\,3} \\ \\ B_{1\,3}B_{1\,3} + B_{2\,3}B_{2\,3} + B_{3\,3}B_{3\,3} \\ \\ B_{1\,j}B_{1\,3} + B_{2\,j}B_{2\,3} + B_{3\,j}B_{3\,3} \end{pmatrix} \qquad j = 3, 4, \cdots, n
$$

$$
4)\quad B_n = \begin{pmatrix} B_{1\,1}B_{1\,n} \\ \\ B_{1\,2}B_{1\,n} + B_{2\,2}B_{2\,n} \\ \\ \vdots \\ \\ B_{1\,n-2}B_{1\,n} + \cdots + B_{n-2\,n-2}B_{n-2\,n} \\ \\ B_{1\,n-1}B_{1\,n} + B_{2\,n-1}B_{2\,n} + \cdots + B_{n-1\,n-1}B_{n-1\,n} \\ \\ B_{1\,n}B_{1\,n} + B_{2\,n}B_{2\,n} + \cdots + B_{n-1\,n-1}B_{n-1\,n-1} + B_{n\,n}B_{n\,n} \end{pmatrix}
$$

We can summarize the provided process in an equivalent process $B = (B_k)\quad k = 1, 2, 3, \cdots, n$ given by:

$$
B = [B_1 B_2 \cdots B_n]\,.
$$

That is steps 1 to n characterize the behavior of the process with a decreasing index j. This is a reconfiguration version of the matrix inverse computation and we then use it to state the following theorem:

Theorem 2.1: The generelized reconfigurable matrix inverse process method can compute full $n \times n$ matrix inverse A and B by using the reconfigurable matrix inverse computations.

The proof idea of this theorem is based on the fact that, if such a process really exist. Then there are two such similar processes that is we can find them by using the reconfigurable matrix inverse computations. The two constructed processes are symmetric. Let denote them by A and B. Combining these two processes comes to the following two cases:

$$
A \cdot B = \begin{cases} V^t \cdot V \cdot R \cdot R^t & \text{if} \quad V = V^t \\ \\ R^t \cdot R \cdot V \cdot V^t & \text{if} \quad R = R^t \end{cases}
$$

Using the state-of-the-art reconfigurable matrix inverse

$$
V \cdot R = R \cdot V = \text{Identity},
$$

then $A \cdot B$ must be the identity. Because the $n \times n$ matrices specified by A and B are full matrices and not upper triangular nor lower triangular. The provided process generalises the reconfigurable matrix inverse computations.

$$
A1 = \begin{bmatrix}
1.0000 & -0.8944 & -0.9621 & -0.9816 & -0.9869 & -0.9886 & -0.9892 \\
-0.8944 & 1.0000 & 0.9058 & 0.9275 & 0.9340 & 0.9363 & 0.9372 \\
-0.9621 & 0.9058 & 1.0000 & 0.9721 & 0.9782 & 0.9802 & 0.9810 \\
-0.9816 & 0.9275 & 0.9721 & 1.0000 & 0.9904 & 0.9923 & 0.9931 \\
-0.9869 & 0.9340 & 0.9782 & 0.9904 & 1.0000 & 0.9960 & 0.9968 \\
-0.9886 & 0.9363 & 0.9802 & 0.9923 & 0.9960 & 1.0000 & 0.9981 \\
-0.9892 & 0.9372 & 0.9810 & 0.9931 & 0.9968 & 0.9981 & 1.0000
\end{bmatrix}
$$

$$
B1 = \begin{bmatrix}
6.6189 & -0.9504 & -0.5100 & -0.2349 & -0.1172 & -0.0648 & -0.0391 \\
-0.9504 & 0.2630 & 0.0567 & 0.0270 & 0.0137 & 0.0076 & 0.0046 \\
-0.5100 & 0.0567 & 0.0823 & 0.0159 & 0.0080 & 0.0045 & 0.0027 \\
-0.2349 & 0.0270 & 0.0159 & 0.0231 & 0.0038 & 0.0021 & 0.0013 \\
-0.1172 & 0.0137 & 0.0080 & 0.0038 & 0.0079 & 0.0011 & 0.0007 \\
-0.0648 & 0.0076 & 0.0045 & 0.0021 & 0.0011 & 0.0032 & 0.0004 \\
-0.0391 & 0.0046 & 0.0027 & 0.0013 & 0.0007 & 0.0004 & 0.0016
\end{bmatrix}
$$

$$
A2 = \begin{bmatrix}
140.0000 & -44.7214 & -51.3459 & -32.8015 & 9.0102 & 77.3491 & 177.1609 \\
-44.7214 & 23.0000 & 19.8375 & 13.3795 & -2.1784 & -28.5140 & -67.9104 \\
-51.3459 & 19.8375 & 47.3538 & 18.1791 & -2.9598 & -38.7429 & -92.2720 \\
-32.8015 & 13.3795 & 18.1791 & 84.1891 & -3.2370 & -42.3707 & -100.9123 \\
9.0102 & -2.1784 & -2.9598 & -3.2370 & 165.7355 & -43.8292 & -104.3860 \\
77.3491 & -28.5140 & -38.7429 & -42.3707 & -43.8292 & 333.6883 & -106.0481 \\
177.1609 & -67.9104 & -92.2720 & -100.9123 & -104.3860 & -106.0481 & 640.5305
\end{bmatrix}
$$

$$
B2 = \begin{bmatrix}
1.0000 & 2.0000 & 3.0000 & 4.0000 & 5.0000 & 6.0000 & 7.0000 \\
2.0000 & 9.0000 & 4.0000 & 5.0000 & 6.0000 & 7.0000 & 8.0000 \\
3.0000 & 4.000025.4000 & 6.0000 & 7.0000 & 8.0000 & 9.0000 & \\
4.0000 & 5.0000 & 6.0000 & 71.3077 & 8.0000 & 9.0000 & 10.0000 \\
5.0000 & 6.0000 & 7.0000 & 8.0000 & 178.0839 & 10.0000 & 11.0000 \\
6.0000 & 7.0000 & 8.0000 & 9.0000 & 10.0000 & 389.2154 & 12.0000 \\
7.0000 & 8.0000 & 9.0000 & 10.0000 & 11.0000 & 12.0000 & 760.4902
\end{bmatrix}
$$

Matrix A1 and B1 and A2 and B2 are computations using the presented extended method. The iteration matrix that was used to achieve the provided inverse is:

$$
A0 = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
2 & 3 & 4 & 5 & 6 & 7 & 8 \\
3 & 4 & 5 & 6 & 7 & 8 & 9 \\
4 & 5 & 6 & 7 & 8 & 9 & 10 \\
5 & 6 & 7 & 8 & 9 & 10 & 11 \\
6 & 7 & 8 & 9 & 10 & 11 & 12 \\
7 & 8 & 9 & 10 & 11 & 12 & 13
\end{bmatrix}
$$

The process computes matrices provided as matrix A and B that are inverse. They are in accordance with conventional matrix inverse results. The implementation is provided in the following listing:

Algorithm 2.2:

```
function COMPLETENESSMETHOD(A, B)
    [m, n] ← size(A)
    [p, q] ← size(B)
    R ← zeros(n, n)
    if m = p or q = n then
        R ← zeros(n, n)
        for j = 2, 3, ··· , n do
            V(:, j) = B(:, j)
            for i = 1 : j − 1 do
                R(i, j) ←
                V(:, i)^t ∗ A(:, j)
                V(:, j) ←
                V(:, j) − R(i, j) ∗ V(:, i)
            end for
            R(j, j) ← ||V(:, j)||
            V(:, j) = V(:,j) / R(j,j)
        end for
    end if
    for j = 1, 2, 3, ··· , n do
        for i = 1, 3, ··· , n do
            B1(i, j) ←
            V(i, j) ∗ V(i, j)^t
            B2(i, j) ←
            V(i, j)^t ∗ V(i, j)^t
        end for
    end for
end function
```

3 DISCUSSION

Some advanced literature has been helpful to conduct our research. We can cite among others

1) From Numerical Recipes in C, the Art of Scientific Computing Second Edition of William H. Press Harvard-Smithsonian Center for Astrophysics and Saul A. Teukolsky Department of Physics, Cornell University and William T. Vetterling Polaroid Corporation and Brian P. Flannery EXXON Research and Engineering Company, available at http://www2.units.it/ipl/students_area/imm2/files/Numerical_Recipes.pdf

2) Algorithms in Matlab [10]

3) Understanding the QR decomposition has been helpful too see [8], as well as the non modified and modified Gram Schmidt orthogonalization method that is closed to the presented implementation although they perform different computations.

Our implementation resulted in the above described architecture. We have used the vectorization features and handled all matrix entries as arrays. So matrix A will be an $n \times n$-size array $A[n][n]$ and matrix B will be an $n \times n$-size array $B[n][n]$. For the description of the reconfigurable matrix inverse please refere to [9]. R2 and R1 are double variable inside which the values of $R[j][k]$ and $R[k][k]$ are accumulated. The vectorization method used has been possible with accumulators R1 and R2 to enable mathematical description transformed into computations. Most of the research connected to this topic use the technique of reconfiguration [11]–[13]. Our analysis although using an algorithm that is deduced from this reconfiguration technique pictures out the implementation and proposes a microarchitecture analysis. The analysis on matrix inversion computation is quite new and to the best of our knowledge, there are no existing standard implementations of these algorithms and no other implementation has been presented so far that will compute the inverse matrix by mean of reconfiguration and the overall execution time of the proposed assembly code provided is 0.0147 seconds. This is the actual standard of the performance of such an implementation. It is difficult to make an objective comparison on how efficient this implementation is since the Recursive Linear Process has not been yet widely applied as new process. The implementation that has been carried is effective and based on reference [9]. We can start with any matrix called the reconfiguration starting matrix and generate through iterations two new matrices that will be inverse. This research is very interesting if we can with the used of the reconfiguration carry many computations and create such new algorithms. The Xilinx hardware implementation has not been yet been conducted. This will be a difficult task since the matrices are not only restricted to integer values. Such an implementation will not be trivial with the Xilinx technology. The prediction of the partial reconfiguration matrix inverse is also very important. One might wish to get two computed inverse matrices, what will be the starting matrix on which we wish to iterate. A general idea has been thought but the implementation has not already been conducted to validate this idea. This concept of reconfiguration combined with the proposed architecture is very important and this implementation can be used as guideline for reconfigurable algorithms. The size of the provided matrix plays no role. The construction of hardware with the Xilinx Technology can be easily carried with the provided analysis but the implementation there will be difficult. One of the main advantages of this implementation is that, no restrictions on the input matrices are set. Any matrix can act as iteration starting matrix and the necessary calculations will be performed.

4 CONCLUSIONS

We have presented the descriptive implementation of the reconfigurable matrix inverse computations and extended the process that can now handle all kind of matrix inverse. Because of the technical approach in this paper, we have proposed a way how to implement the reconfigurable matrix inverse process. Our scientific approach basically intends to connect the concept of reconfiguration to process creation. The classical study of algorithms and computational method see [10], [14]–[21] has been achieved at the same time. This paper provides implementation aspects of the developed process. From the point of view of our analysis, this investigation is general and by this way many matrix computations will be achieved by reconfiguration on a specific matrix. There are concrete applications of our analysis. A few of them are listed in the field of matrix based engineering and mathematical matrix computations. That is iterating on matrices using reconfiguration to achieve computations in a non standard way. Some mathematical backgrounds [20]–[22] have been necessary for the accomplishment of this paper. Our research extends at the meantime conventional way how to handle matrix based computations and analysis. The following books [22]–[27] [28]–[30] have been helpful to the implementation and thus, for translation of the process into codes. We have taken advantage of the ARM technology for the validation of the provided micro-architecture description and the provided assembler listing. Although our implementation is based on the ARM test

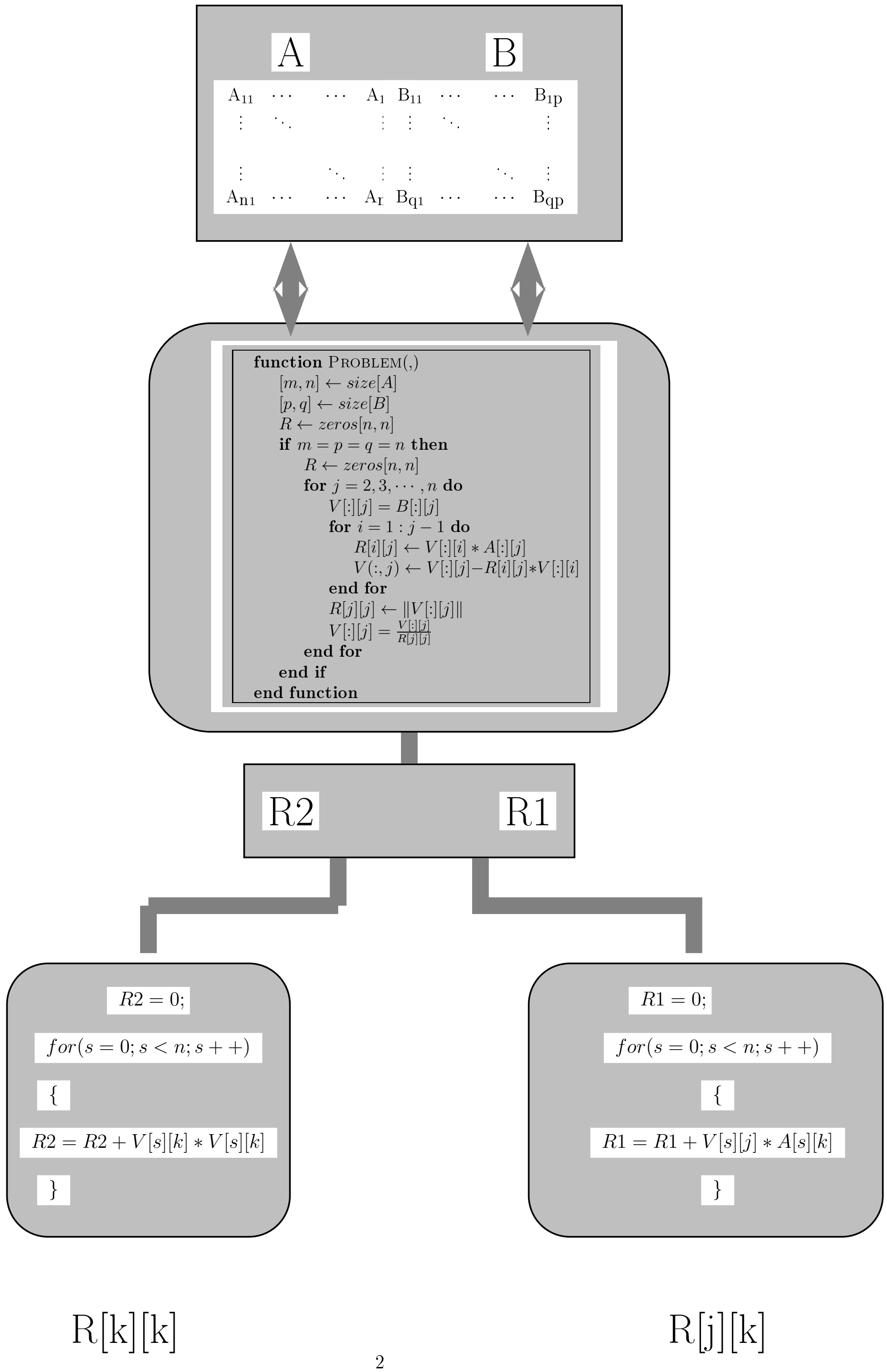

Fig. 2: Reconfigurable Matrix Inverse Implementation the Last two Frames Picture out the Computations of R[k][k] and R[j][k]

environment, the description provided is universal. Due to the technical approach in this paper, we have avoided simulations for since a wide range of matrices has been used to test our implementation. Our concentration was instead drawn on the realization of the process in codes. Listing 1 pictures out a piece of such codes. The approach in this paper is technical and will help many process and algorithm designers and computational engineers.

References

[1] C.Kyrkou and T.Theocharides, "A parallel hardware architecture for real-time object detection with support vector machines," *IEEE Transactions on Computers (TC)*, vol. 61, no. 6, pp. 831–842, 2012.

[2] Xilinx, "Xilinx documentation web site," 2012. [Online]. Available: http://www.xilinx.com/support/documentation/white-papers/wp374-partial-reconfig-xilinx-FPGAs.pdf

[3] E. Mbock, "FPGA Implementation of the Kalman Filter," 2011. [Online]. Available: http://www.dpg-verhandlungen.de

[4] T. T. Andreas G. Savva and V. Soteriou, "Intelligent on/off dynamic link management for on-chip networks," *Journal of Electrical and Computer Engineering*, 2012.

[5] M. D. S. A. Donato, F. Ferrandi and D. Sciuto, "Operating system support for dynamically reconfigurable soc architectures," *IEEE International Soc Conference*, pp. 233–238, 2005.

[6] A.Brzezinski and E.Modiano, "Dynamic reconfiguration and routing algorithms for ip-over-wdm networks with stochastic traffic," *In Proceeding of IEEE Infocom*, 2005.

[7] E. Mbock, "Dynamic Partial Reconfiguration based on the Kalman Filter Method," *Proc. of Iasted on Control and Applications*, 2013.

[8] ——, "Partial Reconfiguration of a Linear Recursive Process and Application on [Q,R]-Decomposition," *The 2013 Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, 2013.

[9] E. M. Mbock, *Algorithm Based Partial Reconfiguration with Application on Matrix Inverse Computations:Transactions on Engineering Technologies*. Springer, 2014.

[10] E. Mbock, *Algorithms in Matlab, the Reality of Abstraction and the Power of Pseudocodes*. Göttingen: Optimus Verlag, 2012.

[11] F. M.D.Santambrogio A.Donato and D.Sciuto, "Operating system support for dynamically reconfigurable soc architecture," *IEEE International Soc Conference*, p. 233238, 2005.

[12] J. Hillman and I.Warren, "Quantitative analysis of dynamic reconfiguration algorithms," *Proc. Int. Conf. Design, Analysis and Simulation of Distributed Systems*, 2004.

[13] Vaidyanathan and J.Trahan, "Dynamic reconfiguration: Architectures and algorithms," *Springer, Berlin, Germany*, 2004.

[14] J. Demmel, *Applied Numerical Linear Algebra*. Philadelphia, pa: Siam, 1997.

[15] G. Golub and C. V. Loan, *Matrix Computations*, 2nd ed. Baltimore, md: Johns Hopkins University press, 1989.

[16] M. A. M. Forsythe, George E. and C. B. Moler, *Computer methods for Mathematical Computations*. Englewood Cliffs, nj: Prentice Hall, 1977.

[17] M. Heath, *Scientific Computing: an Introductory Survey*. Boston, ma: Mcgraw-hill, 1997.

[18] J. F. R.L. Burden, *Numerical analysis*. Brooks/Cole Publishing Company, 1997-2001.

[19] N. Higham, *Accuracy and Stability of Numerical Algorithms*. Philadelphia, pa: Siam, 1996.

[20] M. Heath, *Scientific Computing: an Introductory Survey*, 2nd ed. Boston, ma: Mcgraw-hill, 1997.

[21] N. Higham, *Accuracy and Stability of Numerical Algorithms*. Philadelphia, pa: Siam, 1996.

[22] R. Skeel and J. Keiper, *Elementary numerical Computing with Mathematica*. New York, ny: Mcgraw-hill, 1993.

[23] B. Hahn, *Essential Matlab for Scientists and Engineers*. New York, ny: John Wiley Sons, 1997.

[24] D. Hill and D. Zitarelli, *Linear Algebra Labs with Matlab*, 2nd ed. Upper Saddle River, nj: Prentice Hall, 1996.

[25] R. Larson and B. Edwards, *Elementary Linear Algebra*, 3rd ed. Lexington, ma: D.C.Heath and Company, 1996.

[26] S. Leon, *Linear Algebra with Applications*, 5th ed. Upper Saddle River, nj: Prentice Hall, 1998.

[27] G. Strang, *Linear Algebra and its Applications*, 2nd ed. Orlando, fl: Academic Press, 1980.

[28] Aström, *Introduction to Stochastic Control Theory*. Academic Press, 1970.

[29] R. Borne, *Commande Optimal Control, Techniques Ingenieur, r-7427*, 1976.

[30] Jacobs, *Introduction to Control Theory*, 2nd ed. Oxford University Press, 1993.