ACCELERATING SVD ON RECONFIGURABLE HARDWARE FOR IMAGE DENOISING

A. Ahmedsaid[*], *A. Amira*
School of Computer Science
Queen's University Belfast
United Kingdom
[*]a.ahmedsaid@qub.ac.uk

ABSTRACT

This paper presents the implementation on FPGA of a block SVD method for image denoising. This method exploits the fact that only the smallest singular values are affected by the noise and therefore can be discarded leading to an efficient non linear image filtering. An efficient architecture for Singular Value Decomposition (SVD) based on the Brent, Luk, Van loan (BLV) systolic array has been proposed. The architecture is three times more efficient and three times faster than the existing BLV structure. An optimised implementation has been efficiently carried out on the PP-RC1000 board using a high level language for hardware design "Handel-C".

1. INTRODUCTION

The SVD has been successfully applied to many image restoration problems. Usual applications include linear space invariant and linear space variant pseudoinverse filtering, image enhancement, separation of 2-D filtering operations into 1-D filtering operations, generation of small convolution kernels etc [1]. Among all unitary transformations, SVD is optimal for images in terms of the energy packed in a given number of transformation coefficients is maximised [2]. Although applicable in many image restoration applications, SVD is severely limited because of a large number of computations required for calculating singular values and singular vectors of large image matrices [2].One of the SVD applications is the block SVD image filtering , where the SVD is applied to each block of a partitioned image (figure 1):

A noisy image *Y* can be described as the addition of the original image F with the random noise *R*:

$$Y = F + R \tag{1}$$

The image *Y* is divided into non-overlapping $N \times N$ square blocks *A*. The SVD of each block is computed as follows:

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T \tag{2}$$

Depending on the noise level, only the smallest singular values are contaminated by the noise. Thus the singular values smaller than a threshold ε are discarded. The denoised blocks *B* of *F* can be approximated using the remaining singular values and their corresponding singular vectors:

$$\hat{B} = \sum_{i=1}^{p} \sigma_i u_i v_i^T \qquad \text{where } p < r \tag{3}$$

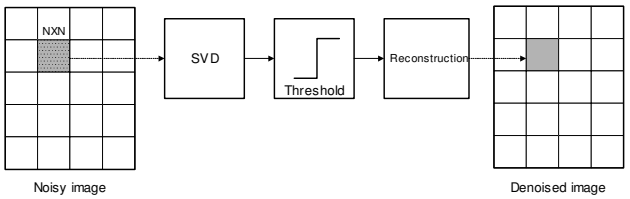

Figure 1. Block SVD image filtering

Because the SVD requires extensive computations, it is the aim of this work to develop an FPGA based SVD architecture for the acceleration of the block SVD image noise filtering. The proposed architecture is a parallel SVD architecture based on the BLV systolic array [3],[4]. The proposed architecture has been implemented and run on the FPGA prototyping board RC-1000 [10]. The implementation results for a block size of 8×8 showed that the FPGA SVD architecture running at 84.44 MHz is almost twice faster than a Pentium 4 processor running at 1.8GHz.

The composition of the rest of the paper is as follows:
A review of the BLV array is given in section 2. The improvement of the BLV array is described in section 3 and in section 4, the FPGA implementation with results are presented. Finally, concluding notes are given in section 5.

2. A REVIEW OF THE BLV ARRAY

The BLV array (figure 2). is a square systolic array that implements a parallel Jacobi [5] SVD/EVD algorithm using $(N/2)^2$ processors (*N* is the matrix size). It performs *N/2* sub-problem in parallel, and a sweep in *N-1* steps. This systolic array is capable of processing the SVD/EVD of a square matrix with a $O(N \log(N))$ time complexity [3],[4].

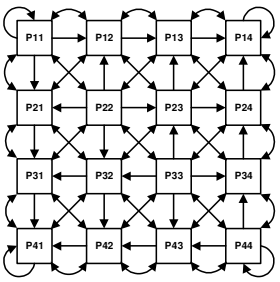

Figure 2. BLV array, N=8. (Horizontal arrows: transmission of left rotation parameter; vertical arrows: transmission of right rotation parameter)

Initially each processor *Pij* holds a 2×2 sub-matrix. A step of the Jacobi algorithm consists of the solution of a 2×2 SVD/EVD problem (computation of a two-sided plan rotation), executed by the diagonal processors and then, the application of the computed rotations to the local 2×2 sub-matrices, executed by all processors. After the application of the rotations, the local matrices are interchanged (figure 2, diagonal connections) between processors for the execution of a new step.

To avoid broadcasting the rotation parameters at constant time along processor rows and columns, the rotation parameters are transmitted at constant speed between adjacent processors. A processor cannot commence a rotation until data from earlier rotations are available on all its lines. Thus, a processor *Pij* stays idle for two time steps while waiting for the processors $P_{i\pm1,\,j\pm1}$ to complete their (possibly delayed) steps. The price paid to avoid broadcasting is that each processor is active for only one third of the total computation [3],[4].

3. IMPROVEMENT OF THE EFFICIENCY OF THE BLV ARRAY

To improve the efficiency of the BLV array we propose to synchronize the processors according to availability of new data instead of an iteration step synchronization strategy [3],[4]:

New algorithm processor

If $i = j$ {diagonal processor}
- Solve 2×2 SVD/EVD
- Output rotation parameters
- Apply rotations to sub-matrix
- Output data (2×2 sub-matrix)
- Wait for new data

Else {off-diagonal processor}
- Wait for new rotation parameters
- Output rotation parameters
- Apply rotations to sub-matrix
- Output data (2×2 sub-matrix)
- Wait for new data

In this new algorithm a processor starts working as soon as new data appears in its input lines. The operations are *data driven* rather than steps synchronized.

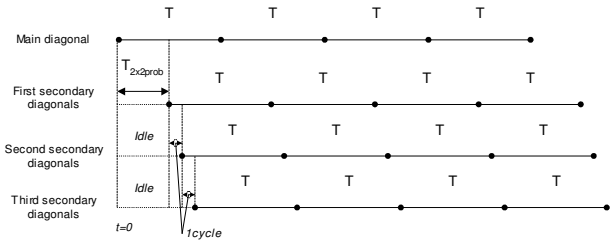

Figure 3. Overlapped processors steps (*T* is the computation time of one step)

Figure 3 shows that all the processors work at every step whereas they stay idle for two steps in the original synchronization method [3],[4]. Further, a step in our algorithm takes $T \approx T_{2\times 2\,prob} + \max(T_{rd}, T_{ro})$, and for the original algorithm $T' \approx \max(T_{2\times 2\,prob} + T_{rd}, T_{ro})$ (where T_{rd} is the time for the rotations application in a diagonal processor, and T_{ro} is the time for the rotations application in an off-diagonal processor). This means that the time of one step in both algorithms is comparable and can be equal (example if $T_{rd} = T_{ro}$). Therefore the new algorithm is about three times more efficient than the precedent one. Another consequence of the new algorithm is the division of the computation time by three: if S is the number of sweeps then the computation time is $CPT = \big(3S(N-1) + (N/2-1) + 3\big)T$ and for the new algorithm $CPT = S(N-1)T + T_{2\times 2\,prob} + N/2 - 1$. That's because in the original algorithm a sweep takes $3(N-1)$ time steps [3],[4] while in our algorithm it takes $(N-1)$ time steps.

4. FPGA IMPLEMENTATION OF THE SYSTOLIC ARRAY

The implementation has been carried out using Handel-C which is a high level language for hardware design based on ANSI C [10]. The implemented systolic array computes both the singular values and singular vectors. The design is completely parametric and platform independent.

4.1 SVD chip

The design comprises a "Control Unit", an "input interface", an "output interface" and the "SVD/EVD array" (figure 4). The control unit generates commands to the processors and handles all the IO operations between the FPGA and the off-chip memory and eventually the host PC. The data matrices are read/written from/to the off-chip memory element by element, which makes the IO requirement independent from the matrix size. The *commands* can be an instruction for IO phase, processing of a new iteration or other commands that can be used in future modifications. The input and output interfaces are basically shift buffers used to transfer data and commands serially from the control unit to the systolic array and output data from the systolic array to the control unit. The CORDIC algorithm has been used for the implementation of plan rotations [7],[8] The solution of a 2×2 sub-problem is performed by the TPR method [9]. The diagonal processors have two CORDIC angle solvers in the case of the SVD and only one for the EVD. All the processors have CORDIC rotation module for the application of the plan rotations.

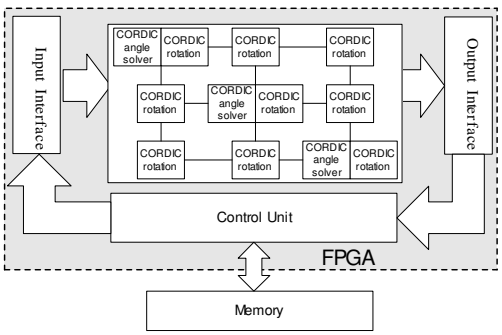

Figure 4. FPGA implementation of the SVD/EVD array

4.2 Computation time and efficiency

The computation time for one step is:

$$T = 23W + T_{sc} + T_e + 111 \quad (7)$$

Where W is the word length, T_e the time for matrices interchange between off-diagonal processors, and is equal to one if the matrix size is 4 else it is equal to two. T_{sc} is the latency of the scale factor (CORDIC has a gain of 1.646) correction module. The constant 111 is due to the initialisation clock cycles which are repeated many times.

A diagonal processor stays idle for $(8W + 40)$ clock cycles whereas an off-diagonal processor stays idle for $(3W + 3)$ clock cycles. If we define the efficiency as the time a processor actually runs during a step then:

$$\rho_{dp} = \frac{T - (8W + 40)}{T} = \frac{15W + T_{sc} + T_e + 71}{23W + T_{sc} + T_e + 111} \quad (8)$$

$$\rho_{op} = \frac{T - (3W + 3)}{T} = \frac{20W + T_{sc} + T_e + 108}{23W + T_{sc} + T_e + 111} \quad (9)$$

where ρ_{dp} and ρ_{op} are respectively the efficiencies of diagonal and off-diagonal processors. The efficiency of the processors is almost constant and its minimal value is 65.6 % ($\approx 15W/23W$) for a diagonal processor and 88.5 % ($\approx 20W/23W$) for an off-diagonal processor.

The number of diagonal processors is $N/2$ and the number of off-diagonal processors is $(N/2)^2 - N/2$, so the average efficiency for the entire array is:

$$\rho_a = \frac{\left((N/2)^2 - N/2\right)\rho_{op} + \left(A_{dp}/A_{op}\right)(N/2)\rho_{dp}}{\left((N/2)^2 + \left(A_{dp}/A_{op} - 1\right)(N/2)\right)} \quad (10)$$

where A_{dp} is the area consumed by a diagonal processor, and A_{op} is the area consumed by an off-diagonal processor. $A_{dp} \approx 3A_{op}$ as the diagonal processors contain three CORDIC modules while the rest of the processors have just one.

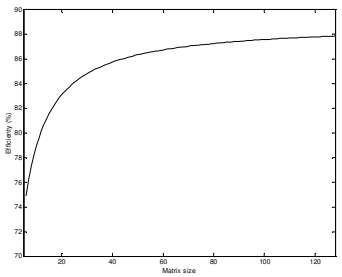

Figure 5. Efficiency of the SVD array (W=24)

Figure 5 shows that the efficiency increases with the matrix size from 75 % to reach its limit ρ_{op}. Note that if such a rigorous computation is applied to the original BLV array, the efficiency would be less than 33% as a processor may stay idle for some time during a working step.

4.3 Implementation results

Table 1,2 shows implementation results for a Xilinx XCV2000E-6 FPGA target [11].

Table 1: Design statistics for matrix size of 6×6

Word Length	Maximum Frequency (MHz)	Area (slices)	Area (%)
14	96.15	9911	51
16	88.54	11297	58
18	84.77	12933	65
20	94.88	13923	72
22	85.40	15141	78
24	87.24	16374	85
26	83.98	17633	91

The maximum frequency is fluctuating around 88 MHz and the area varies linearly with the word length. This is due to the fact that multipliers (which have long delays and an $O(w^2)$ area complexity) haven't been used. It is one of advantages of the CORDIC algorithm which avoids us using expensive and complex multipliers, dividers and square rooters

Table 2: Influence of the matrix size

Matrix size		4	6	8
Word length				
14	Max Freq (MHz)	87.73	96.15	85.98
	Area (slices)	4827	9911	16681
	Area (%)	25	51	86
16	Max Freq (MHz)	96.38	88.54	84.44
	Area (slices)	5507	11296	19013
	Area (%)	28	58	99

The area increases quickly with matrix size. That's because the array uses $(N/2)^2$ processors.

4.4 Application to image denoising

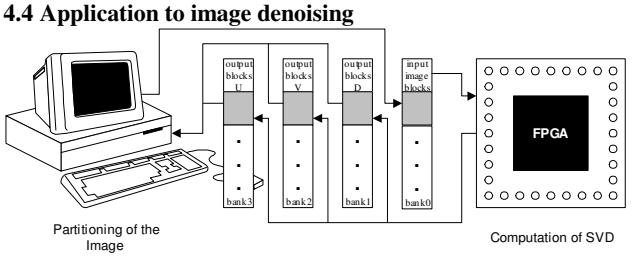

Figure 6. Implementation of the denoising application

A host programme is used to open images, partition them and store them in the RC1000 first memory bank [10]. The FPGA computes the SVD of all blocks and stores the results in the remaining memory banks (figure 6). The host PC reconstructs the image blocks according to equation (3). The result of the SVD filtering for a block size of 8×8 and a world length of 16 bits applied to the cameraman image (256×256) is shown on figure 7.

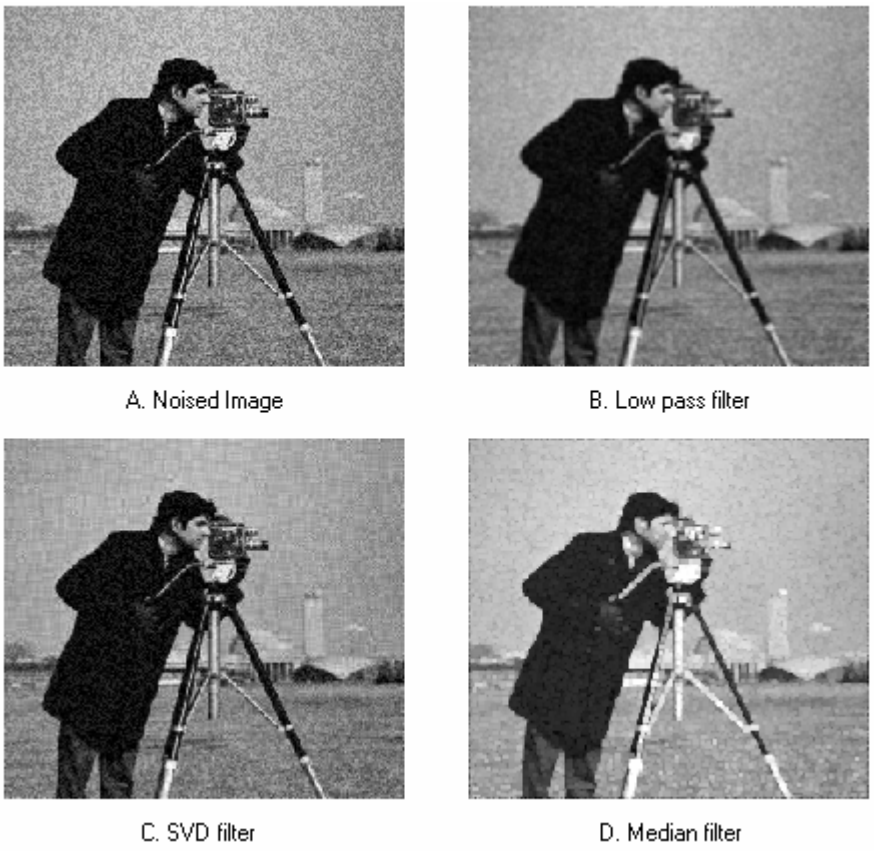

Figure 7. Block SVD image filtering (*N*=8,*W*=16,*SNR*=11db)

The SNR of the image *B* is 12 db while it is 14 db for the image *C*. An improvement of 3 db has been achieved by this technique which removes the noise without blurring the image like an ordinary low pass. For these specific parameters the computation time of the SVD of one block is 0.123ms (fmax=84.44 MHz) which means 8130 blocks per second can processed. A test of the Matlab SVD function (which uses a faster algorithm) on a Pentium 4 running at 1.8GHz showed that ,in average , 4200 SVDs of 8×8 can be processed per second. Although the FPGA is running at 84 MHz it is almost twice (1.9) faster than a Pentium4 running at 1.8 GHz. Further, the time complexity is $O(N\log N)$ for this array with a large constant and $O(N^3)$ for the standard SVD algorithm, which makes the array competitive only for large matrices, and yet for a 8×8 matrix a good performance is achieved.

5. CONCLUSION

The block SVD noise filtering is a method which has good denoising capabilities without introducing image blurring. In order to accelerate this method, an improved architecture for the BLV array which led to an improvement of the efficiency by three times and the division of the computation time by three has been proposed. This novel architecture has been efficiently implemented on FPGA using a High-level language. The design is parametric, flexible and can be adapted to the application's requirements.

6. REFERENCES

[1] H.C. Andrews, C.L. Patterson,'Singular values and digital image processing', IEEE trans ASSP, Vol. 24, pp. 26- 53.

[2] Z. Devcic and S. Loncaric. SVD Block Processing for Non-linear Image Noise Filtering. Journal of Computing and Information Technology, Vol. 7, No. 3, pp. 255-259, 1999.

[3] R.P. Brent, and F.T. Luk, "The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays" SIAM J. SCI. STAT. COMPUT, vol. 6, no. 1, pp. 69-84, January. 1985.

[4] R.P. Brent, F.T. Luk, and C. Van Loan "Computation of singular value decomposition using mesh-connected processors" J. VLSI. Comput Syst, vol. 1, no. 3, pp. 242-270, 1985.

[5] G.H. Golub, and C.F. Van Loan, "Matrix computations", The Johns Hopkins University Press, London, 1996.

[6] A. Amedsaid, A. Amira, A. Bouridane, "Improved SVD systolic array and implementation on FPGA", (FPT'03).

[7] J.Volder, "The CORDIC Computing Technique", IRE Trans. Comput., Sept. 1959, pp.330-334.

[8] J.R. Cavallaro, F.T. Luk, "CORDIC arithmetic for an SVD processor" Journal of parallel and distributed computing, vol. 5, pp. 271-290, 1988.

[9] B. Yang, and J.F. Bohme, "Reducing the computations of the singular value decomposition array given by Brent and Luk" SIAM J. Matrix. Anal. Appl, vol. 12, no. 4, pp. 713-725, October. 1991.

[10] www.Celoxica.com

[11] www.Xinlinx.com