POLITECNICO DI TORINO
Repository ISTITUZIONALE

Efficient FPGA Implementation of PCA Algorithm for Large Data using High Level Synthesis

Original
Efficient FPGA Implementation of PCA Algorithm for Large Data using High Level Synthesis / Mansoori, Mohammadamir; Casu, Mario R.. - ELETTRONICO. - (2019), pp. 65-68. (Intervento presentato al convegno 2019 15th Conference on Ph.D Research in Microelectronics and Electronics (PRIME) tenutosi a Lausanne (CH) nel 15-18 July 2019) [10.1109/PRIME.2019.8787782].

Availability:
This version is available at: 11583/2748941 since: 2019-08-28T20:05:16Z

Publisher:
IEEE

Published
DOI:10.1109/PRIME.2019.8787782

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

05 April 2024

Efficient FPGA Implementation of PCA Algorithm for Large Data using High Level Synthesis

Mohammad Amir Mansoori, Mario R. Casu
Department of Electronics and Telecommunications, Politecnico di Torino, Turin, Italy.
e-mail: {*mohammadamir.mansoori, mario.casu*}*@polito.it*

***Abstract*—Principal Component Analysis (PCA) is a widely used method for dimensionality reduction in different application areas, including microwave imaging where the size of input data is large. Despite its popularity, one of the difficulties in using PCA is its high computational complexity, especially for large dimensional data. In recent years several FPGA implementations have been proposed to accelerate PCA computation. However, most of them use manual RTL design, which requires more time for design and development. In this paper, we propose an FPGA implementation of PCA using High Level Synthesis (HLS), which allows us to explore the design space more efficiently than with hand-coded RTL design. Starting from a PCA algorithm written in C++, we apply various hardware optimization techniques to the same code using Vivado HLS in order to quickly explore the design space. Our experiments show that the performance of the design obtained with the proposed method is superior to the state-of-the-art RTL design in terms of resource utilization, latency and frequency.**

***Index Terms*—PCA, Hardware optimization, FPGA, Vivado HLS, Large data.**

Principal Component Analysis (PCA) is a mathematical method used to transform a set of variables into a lower dimensional data preserving the most representative information. It is often used in machine learning and image processing as the first step for feature extraction and representation. We are particularly interested in the application of PCA to biomedical images obtained with Microwave Imaging (MI), where we deal with large dimensional images. For example, it can be used for breast tumor classification [1] or feature extraction [2] to reduce data dimensions.

The complexity of PCA calls for efficient hardware implementations capable of handling large data sets. Field Programmable Gate Arrays (FPGAs) are suitable for this purpose, due to their inherent parallelism. Several FPGA-based methods have been proposed to accelerate PCA computations. One of the most time-consuming parts of PCA is the computation of the eigenvalues of the covariance matrix of the input data, which has been the subject of considerable efforts for hardware acceleration. Bravo et al. presented a hardware architecture for eigenvector decomposition of a covariance matrix based on the Jacobi method [3]. Using two CORDIC modules and a Finite State Machine (FSM), they could improve the accuracy of eigenvector computation. Ledesma et al. presented an FPGA implementation of Singular Value Decomposition (SVD) for relatively large matrices up to the size of 32×127 [4]. A new Register-Transfer Level (RTL) method for the design of PCA algorithm was proposed in [5] for object detection.

Recently, there has been an increasing tendency towards using High Level Synthesis (HLS) rather than RTL in the design of FPGA hardware. HLS reduces significantly the development time and allows designers to quickly explore a large design space in the quest for the best solutions. For example, in [6], a hardware accelerator obtained with HLS was proposed for the application of fall detection systems using PCA as an internal stage. However, the PCA components were computed in software and used in the final hardware computation inside the FPGA. In [7], a comparative study between various RTL and HLS designs of the Jacobi algorithm enabled the designers to determine the best hardware implementation.

In this paper, we propose a new FPGA architecture for the implementation of the PCA algorithm using HLS, which we show being an efficient solution when dealing with large input data. Previous methods either do not use HLS, or do not implement the whole algorithm, or consider an off-line stage for the computation of PCA coefficients. In addition, the size of input matrices in previous works is relatively small compared to the large dimensional data used in our target MI applications. Thanks to HLS, we could validate in a short time span various hardware optimization techniques applied to the same high-level code and converge rather quickly to the final solution with the best performance in terms of execution time.

To validate our HLS approach, we looked for existing RTL implementations to be used as reference designs. To the best of our knowledge, however, there are no RTL designs for PCA specifically aimed at MI applications. Therefore, we had to select from the literature a state-of-the-art RTL implementation for a different application with similar characteristics. We then used as reference design an RTL implementation of PCA for hyperspectral imaging [8]. In hyperspectral imaging, images are obtained in many different bands in the microwave spectrum. Usually, there is redundant information in different spectral bands, which can be removed using PCA.

In the following, a brief overview of PCA is presented and the proposed hardware is described. Finally the results in terms of performance and utilization of FPGA resources are reported.

I. Proposed Method

A. Overview of PCA algorithm

The inputs to the PCA algorithm are a matrix $X_{N\times B}$ and its covariance $C_{B\times B} = X^t \times X$, in which N are the pixels in each image and B are the bands. Then, the algorithm goes through the following steps to obtain the output matrix $Y_{N\times L}$ represented on a smaller number of dimensions ($L < B$):

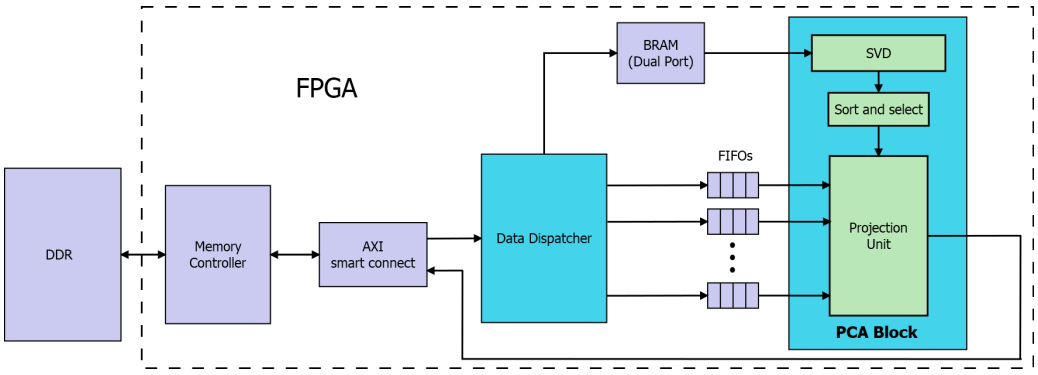

Fig. 1. Block diagram of the proposed PCA implementation. HLS blocks are in blue and the other color is for Vivado IPs.

1) SVD of the covariance matrix: $C = U\Sigma U^T$;
2) Sorting singular values and vectors in descending order: $\Sigma^s, U^s = Sort(\Sigma, U)$;
3) Selection of principal components based on the cumulative energy of singular values: $E_{B\times L} = Select(\Sigma^s, U^s)$;
4) Projection of input data into the new base: $Y = X \times E$.

In order to select the number of principle components, the following procedure is used. At first, the total energy TE of the sorted singular values is computed[1]:

$$TE = \sum_{i=1}^{N} \sigma_i, \tag{1}$$

where $\sigma_i \in \Sigma^s$ is the ith component of the sorted vector of eigenvalues. Since most of the information is stored in the first few components of Σ^s, only $L < B$ principal components can be selected. This number is determined by setting a proper threshold Θ (%) as a percentage of the total energy and then by choosing the first L components for which the following criteria is met:

$$100 \times \frac{\sum_{i=1}^{L} \sigma_i}{TE} \geq \Theta \tag{2}$$

We used two different data sets to validate our design, which were also used in the reference [8]. The number of spectral bands is $B = 224$ for both data sets, but the image dimension for the first one is N=350×350 and for the second one is N=614×512. This clearly shows the large data dimensions used in this application, especially in the second data set.

For these data sets, the number of principal components is $L \leq L_{max} = 24$.

B. Hardware architecture

Our architecture implements steps 1-4 of the PCA algorithm outlined above. It does not include the computation of the covariance matrix for a fair comparison with the reference.

The hardware architecture is shown in Fig. 1. We designed the two main blocks "Data Dispatcher" and "PCA block" in HLS, whereas the other parts are used for data storage and communication and are existing Intellectual Property (IP) blocks provided by Vivado, the Xilinx FPGA tool that we use in our design.

A DDR memory is used to store the input image pixels and the covariance matrix as well as the output of the PCA block. The Memory Controller is in charge of reading and writing data from and to the memory. It uses the AXI smart connect block to share one AXI port for both writing to the Data Dispatcher and reading from the PCA block.

The Data Dispatcher receives first the covariance matrix and sends it to the BRAM; then it receives the input pixels from diffrent spectral bands and sends them to the FIFOs.

While the pixels are transferred to the FIFOs, the PCA block reads the covariance matrix from the BRAM, performs SVD calculation, and sorts and selects the principal components. Such concurrency of SVD computation and pixel transfer allows us to significantly reduce the overall latency. Finally, the input data are projected into the new base, resulting in lower dimensional output data.

C. Design of the PCA block

To compute the singular values of the covariance matrix, we started from an existing SVD code available in Vivado HLS, the Xilinx tool that automatically generates logic-synthesizable RTL from a high-level description. To this code, we applied different optimization directives, which are described in the next subsection. The computed singular values are sorted in descending order and then the principal components are selected based on (1) and (2). Note that in HLS we had to set a maximum limit on the number of principal components in addition to the energy threshold. As a result, we selected $L_{max} = 24$ based on the analysis in [8]. In addition, the threshold is selected as $\Theta = 98\%$.

In the Projection Unit (PU), the input data matrix of image pixels is multiplied by the matrix of principal components to remove the redundant information from the input images. This is one of the critical parts of the design due to the large number of input pixels. Incorporation of FIFOs into the design is to address the problem of large data communication, to let the transfer of input data matrix occur at the same time with SVD computation, and to improve the execution time of the PU as shown below. Since these are dual-clock FIFOs, we could have

[1]Note that the dimension of the eigenvalues of a covariance matrix is energy, hence their sum is also an energy.

the Data Dispatcher run at a higher clock frequency than the PCA block, which allowed us to speed up the data transfer. The same is true for the Dual-Port BRAM used to store the covariance matrix, in which the two ports are associated to two different clock domains.

Note that all of the parts inside the PCA block are described as a single HLS code and partitioning it into sub-blocks is just for clarification.

D. Hardware optimization techniques

The initial unoptimized RTL generated by HLS was inefficient in terms of latency and resource utilization. To enhance the performance of the design, we applied different optimization methods to the same code.

The most critical parts of the design in terms of latency were the PU and the SVD block. The PU code consists of the three nested loops shown in *Algorithm 1*. These loops iterate on the image pixels (N), the maximum number of selected eigenvalues (L_{max}), and the spectral bands (B), respectively, resulting in a large number of iterations.

Algorithm 1

```
Projection_Loop:
for(int p=0;p<N;p++){
    COLB: for(int c=0;c<L_max;c++){
        #pragma HLS PIPELINE II=4
        tmp=0;
        ...
        COLA: for(int n=0;n<B;n++){
                tmp+=Din_pixel[n]*PC[n][c];}
        Data_Transformed[p][c]=tmp;}}
```

In the above code, `Din_pixel` is a vector to store all the bands for each pixel of the input data and `PC` is the matrix of principal components.

The first optimization method was to pipeline the middle loop so that the resulting hardware can overlap the computation of the projection over different components.

To further reduce the loop Initiation Interval (II), and so to increase the pipeline throughput, the inner loop has to be at least partially unrolled, so that different multiplications and additions can be computed in parallel. This in turn leads to the need to access multiple bands in parallel. Therefore, as a second optimization, we partitioned `Din_pixel` and replaced the default dual-port BRAM that Vivado HLS instantiates with multiple FIFOs as shown in Fig. 1. With 56 partitions and therefore 56 FIFOs we obtained II=4 for the middle loop[2].

The final limitation was inside the SVD function (code not reported for space limitations). This function computes the singular values in an iterative manner in which the diagonal and off-diagonal elements of the matrix are updated in each iteration. The computation of off-diagonal elements accounts for most of the latency. Therefore, the next optimization was to pipeline the loop on the off-diagonal pixels with a minimum initiation interval. For further enhancement, loop unrolling was also applied to this loop.

[2]Since $B = 224$, with 56 partitions we need 224/56=4 clock cycles to compute one pixel for all the principal components.

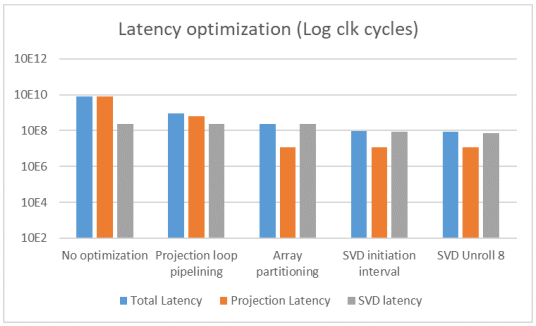

Fig. 2. Results of the HLS procedure for latency optimization.

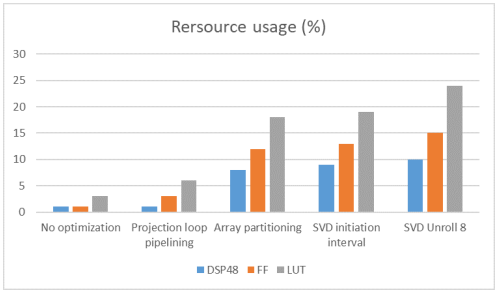

Fig. 3. Effect of HLS optimizations on resource utilization.

As a final note on the hardware design, HLS facilitated the design of all the I/O interfaces including AXI ports, FIFOs and BRAM interfaces, which were obtained without any changes in the code except for adding specific interface directives.

II. Results

We evaluate the quality of our results by comparing resource usage and execution time of our solution with those of the reference RTL implementation targeting the same FPGA, i.e. the Virtex7 of the VC709 evaluation board.

Details about the hardware architecture are as follows. The maximum frequency of the PCA block is 87 MHz whereas for the Data Dispatcher, it is 400 MHz. These two frequencies are generated by a PLL inside the Memory Controller. The width of AXI bus of the memory controller is 512 bits enabling it to provide 16 pixels at every clock cycle for the Data Dispatcher. With this clock rate and bit-level parallelism we can feed the 56 FIFOs without exceeding the DDR access bandwidth.

An important feature of the proposed method is the use of floating point arithmetic, in contrast with the reference RTL design that uses integer arithmetic. This leads to a better accuracy in the final PCA computations.

Fig. 2 shows the results of the optimization steps outlined above. The initial design with no optimization directives has the largest latency. Applying the loop pipelining to the projection unit significantly reduces the latency and array partitioning on the input data further improves it. Finally, reducing SVD initiation interval and using an unroll factor equal to 8 result in the minimum achievable latency.

Fig. 3 shows the effect of different incremental optimizations on the resource utilization. Each additional step led to an

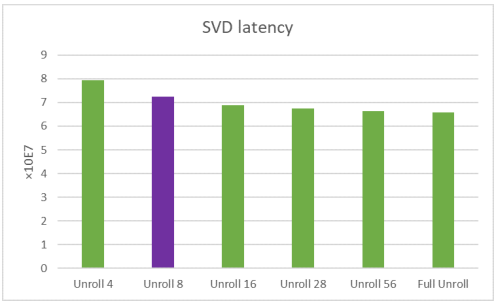

Fig. 4. Effects of unrolling factor on SVD latency in terms of clock cycles.

Fig. 5. Comparison of execution time for PCA algorithm between RTL [8] and HLS (this work).

increased hardware usage. The number of BRAMs is almost constant (44% or 45%), therefore it is not reported.

To select the optimal value of the loop unrolling factor for SVD, we evaluated its effect on both latency and resource usage. Fig. 4 shows that the latency decreases by increasing the unroll factor. However, the Place-and-Route implementation step in Vivado failed for unroll factor 56 and for full unrolling, due to wire congestion errors. Factors 16 and 28 also created timing issues at the target clock frequency, due to high resource usage for these factors. Unrolling by factor 8 gives the best results in terms of both timing and hardware usage.

The final latency for each part of the PCA block is shown in Table I for both datasets. As seen in the table, most of the latency is due to the SVD function.

TABLE I
MAJOR LATENCIES IN THE DESIGN (CLOCK CYCLES).

	Total Latency	SVD latency	Projection latency
Data1	84,432,350	72,414,576	11,760,906
Data2	102,851,653	72,401,196	30,180,233

After place and route in Vivado, the final working frequency, resource utilization, and execution time were obtained for each dataset. Table II and Fig. 5 report these results and compare them with those of the RTL implementation described in [8]. In addition, the maximum power consumption in the design is 9 Watts.

The comparison clearly show that our results are superior in terms of performance as well as resource utilization, except for a slight increase in the use of flip-flop (FF) resources (+4.9%).

TABLE II
COMPARISON OF RESOURCE USAGE BETWEEN RTL [8] AND HLS (THIS WORK).

	BRAM	DSP48	FF	LUT	freq(MHz)
HLS data1	40%	10.2%	11.5%	29.4%	87.67
HLS data2	40%	10.2%	10.6%	29.7%	84.21
RTL	61%	71.6%	6.6%	68%	76

III. CONCLUSIONS

In this paper, a new FPGA hardware architecture is presented for the implementation of PCA algorithm using HLS. Compared to a conventional RTL design, the proposed hardware improves both latency and hardware usage. A maximum of 3.4x speedup is achieved and the hardware utilization is significantly reduced, especially DSP and BRAM resources. The computations in the HLS design use floating point arithmetic rather than integer numbers in RTL, resulting in a better accuracy. In addition, the maximum achievable frequency of the PCA block is about 15% more than the RTL design. These features make the suggested hardware suitable for large dimensional data inputs.

ACKNOWLEDGMENT

This work was supported by the EMERALD project funded by the European Unions Horizon 2020 research and innovation programme under the Marie Skodowska-Curie grant agreement No. 764479.

REFERENCES

[1] S. K. Davis, B. V. Veen, S. C. Hagness, and F. Kelcz, "Breast tumor characterization based on ultrawideband microwave backscatter," IEEE TRANS. Biomedical Engineering, vol. 55, no. 1, pp. 237–246, 2008.

[2] B. Oliveira et al., "Avoiding unnecessary breast biopsies: clinically informed 3D breast tumour models for microwave imaging applications,"IEEE Antennas and Propagation Society International Symposium (APSURSI), pp. 1143–1144, 2014.

[3] I. Bravo et al., "Novel HW architecture based on FPGAs oriented to solve the eigen problems," IEEE Trans. VLSI Systems, vol. 16, no. 12, pp. 1722–1725, 2008.

[4] Luis M. Ledesma-Carrillo et al., "Reconfigurable FPGA-based unit for singular value decomposition of large $m \times n$ matrices," Int. Conf. Reconf. Comp. & FPGAs, pp. 345-350, 2011.

[5] I. Bravo et al., "An intelligent architecture based on field programmable gate arrays designed to detect moving objects by using principal component analysis," Sensors, vol. 10, no. 10, pp. 9232–9251, 2010.

[6] A. Ali, M. Siupik, A. Amira, F. Bensaali, and P. Higuera, "HLS based hardware acceleration on the Zynq SoC: a case study for fall detection system,", 11th Int. Conf. Comp. Sys. and Appl. (AICCSA), pp. 685–690, 2014.

[7] I. Bravo, C. Vazquez, A. Gardel, J. L. Lazaro, and E. Palomar, "High level synthesis FPGA implementation of the jacobi algorithm to solve the eigen problem," Mathematical Problems in Engineering, vol. 2015, Article ID 870569, 11 pages, 2015.

[8] D. Fernandez, C. Gonzalez, D. Mozos, and S. Lopez, "FPGA implementation of the principal component analysis algorithm for dimensionality reduction of hyperspectral images," J. Real-Time Image Proc., pp. 1–12, 2016