2014 IEEE 22nd International Symposium on Field-Programmable Custom Computing Machines

An Efficient Architecture for Floating-Point Eigenvalue Decomposition

Xinying Wang and Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa, USA
Email: {xinying, zambreno}@iastate.edu

***Abstract*—Eigenvalue decomposition (EVD) is a widely-used factorization tool to perform principal component analysis, and has been employed for dimensionality reduction and pattern recognition in many scientific and engineering applications, such as image processing, text mining and wireless communications. EVD is considered computationally expensive, and as software implementations have not been able to meet the performance requirements of many real-time applications, the use of reconfigurable computing technology has shown promise in accelerating this type of computation. In this paper, we present an efficient FPGA-based double-precision floating-point architecture for EVD, which can efficiently analyze large-scale matrices. Our experimental results using an FPGA-based hybrid acceleration system indicate the efficiency of our novel array architecture, with dimension-dependent speedups over an optimized software implementation that range from $1.5\times$ to $15.45\times$ in terms of computation time.**

I. Introduction

Eigenvalue decomposition (EVD) has been widely used as a factorization tool to conduct principal component analysis in many scientific and engineering applications, such as image processing, acoustic processing, mobile communication and remote sensing. To minimize the "dimensionality curse", which refers to the difficulties in managing and analyzing high-dimensional data, EVD can be employed to identify key patterns in the data, after which the original datasets can be approximated with fewer dimensions without losing significant information. In many signal processing applications, EVD is performed iteratively, which incurs a relatively high computational cost for the entire system. As data dimensionality is continuing to increase in scientific and engineering applications, EVD runtime is likely to keep pace.

Eigenvalue decomposition is characterized as the process of orthogonal transformations to diagonalize symmetric matrices, in which large amounts of highly data-dependent rotations are performed iteratively. Efficient software implementations such as MATLAB and LAPACK employ the Householder transformation [1] to diagonalize matrices, which consists of recursive bidiagonalization process and implicit *QR* decompositions; however, the high data dependency and inherent computational complexity of $O(n^3)$ restrict its performance, especially for applications involving large-scale matrices. The recent emergence of Graphic Processing Units (GPUs) in the high performance computing community has allowed for new methods to accelerate many general-purpose computations. However, the multi-dimensional threading structure of GPU computing is not highly compatible with the iterative thread synchronization and irregular memory access required for better EVD convergence making the optimization of these designs on GPUs quite challenging, especially for input matrices with dimensions smaller than 1000 [2], [3].

Modern FPGAs are highly parallel and specialized computational fabrics, and previously researchers have investigated accelerating both EVD and singular value decomposition (SVD) using FPGAs [4], [5]. However, the logic capacity of FPGAs has typically limited the scalability of the adapted matrices [6]–[8], even though this previous work targeted applications in real-time signal processing using fixed-point arithmetic, for which hardware resource utilization is significantly less than for floating-point arithmetic.

In this paper, we present a novel and efficient FPGA-based architecture for eigenvalue decomposition, which attempts to analyze considerably larger matrices than those applied to previous hardware designs, using matrix partition and a pipelined 1D systolic array. Our single FPGA-based design supports double precision float-point operands, offering a wider dynamic range than previous fixed-point implementations. Our experimental results demonstrate the better efficiency of our system compared to optimized CPU-based software solutions, the latest FPGA design for large matrices [8], and a GPU-based implementation when the matrix size is under 2000×2000 [2].

II. Theoretical Background

A. Singular Value / Eigenvalue Decomposition (SVD/EVD)

The singular value decomposition of an $m \times n$ matrix *A* is in the form of eq. (1)

$$A = U\Sigma V' \tag{1}$$

where *U* is an $m \times m$ matrix and *V* is an $n \times n$ matrix, both of which are orthogonal matrices such that $U' \cdot U = V' \cdot V = I$. Σ is an $m \times n$ diagonal matrix with the nonnegative diagonal elements, which are the singular values. Factorization is called EVD when *A* is a squared symmetric matrix.

B. Jacobi Rotations

Jacobi rotations are performed iteratively for matrix diagonalization by using Jacobi rotation matrices J^l and J^r as shown in eq. (2).

978-1-4799-5111-6/14 $31.00 © 2014 IEEE
DOI 10.1109/.25

IEEE computer society

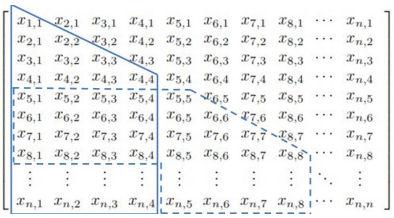

Fig. 1: An example matrix partition for EVD.

$$J^{l'} \cdot \begin{pmatrix} A_{pp} & A_{pq} \\ A_{qp} & A_{qq} \end{pmatrix} \cdot J^{r} = \begin{pmatrix} A''_{pp} & 0 \\ 0 & A''_{qq} \end{pmatrix} \quad (2)$$

The Jacobi matrices J^l and J^r can be obtained through the determinations of plain rotation angles with paired diagonal elements and their respective off-diagonal elements.

III. Related Work

Jacobi-related approaches for eigenvalue decomposition and singular value decomposition, including the two-sided Jacobi Rotation algorithm [4] and the one-sided Jacobi Rotation algorithm [5], provide an opportunity for fine-grain parallelism. Previously, FPGAs were employed to demonstrate the highly parallel implementations of EVD and SVD based on two-sided Jacobi Rotations, by accelerating their independent 2×2 rotations, using a parallel architecture featuring a 2-dimensional systolic array. In this earlier work, the scalability of the applicable matrices had been severely restricted by the limited resources on FPGAs [4], [6], [7]. The Hestenes-Jacobi Method, which is also known as one-sided Jacobi rotation, provides a better opportunity for vectorized parallel operations [3]. However, its architectural design with iterative and repetitive processing limited the overall speedup [3], while GPU implementations have suffered from the overhead associated with thread synchronization and global memory reads [8].

IV. The Partitioned EVD Computation Algorithm

Our partitioned EVD computation algorithm was derived from the two-sided Jacobi approach [4] to zero out all the off-diagonal elements iteratively. The rotations for a symmetric matrix are identical on both sides, whose computations can be reduced by half as the processing on a lower or upper triangular matrix. To improve the scalability of the design, the matrix is first partitioned into a series of vector-blocks and then followed by recursively rotating diagonal elements to annihilate off-diagonals. Each partition consists of numerical diagonal elements and their respective rows and columns in the lower or upper triangular part of the matrix. A partition example is shown in Fig. 1, and vector-block partitions related to the diagonal elements of $x_{1,1}$-$x_{4,4}$ and $x_{5,5}$-$x_{n,n}$ are highlighted by the polygons with solid and dashed lines, respectively. In this paper, for each partitioned vector-block, the diagonal elements and off-diagonal elements in this partition are referred to as *host diagonal* elements and *host off-diagonal* elements respectively, while the remainder

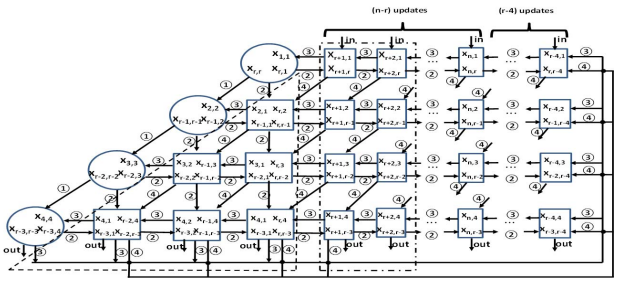

Fig. 2: Demonstration of partitioned Jacobi Rotation approach (for $r \geq 9$).

of diagonal elements and off-diagonal elements in the matrix are referred to as *guest diagonal* elements and *guest off-diagonal* elements respectively. For example, considering the solid lines highlighted partition in Fig. 1, $x_{1,1}$-$x_{4,4}$ are host diagonal elements and $x_{5,5}$-$x_{8,8}$ are guest diagonal elements, while the off-diagonal elements in the first four columns are host off-diagonal elements and the rest of off-diagonals are guest off-diagonal elements. At runtime, the partitioned vector-blocks are processed successively; in processing each partition, host diagonal elements are paired with every other diagonal elements of the matrix to perform Jacobi rotations to zero out all the host off-diagonal elements. Meanwhile, the updates of affected off-diagonal elements are calculated.

V. The EVD architecture

To parallelize our partitioned EVD approach, each partition is mapped to a systolic array of computational processing elements (PEs). Figure 2 shows the example of mapping the partition, which is highlighted by polygon with the solid line in Fig. 1, to the systolic array. As shown in Fig. 2, this systolic array consists of three types of computational PEs: *diagonal Jacobi Rotation* elements, *off-diagonal single update* elements, *off-diagonal double update* elements, in which the diagonal PEs (shown as ovals in Fig. 2) are employed to conduct Jacobi Rotation, while the Off-diagonal Single Update elements and the Off-diagonal Double Update elements are used to update off-diagonal vectors affected by one or two rotations respectively (shown as rectangles in Fig. 2).

Rotation angle parameters *cos* and *sin* are generated by the diagonal PEs and then broadcast to the respective off-diagonal PEs, which are in the same rows and columns with diagonal PEs, to update the remaining elements. Off-diagonal PEs with two off-diagonal elements, are affected only by the rotations in the same row that a "single update" is needed over the two off-diagonal elements; on the other hand, off-diagonal PEs, with which four off-diagonal elements are included, have to update twice on different combinations of the two off-diagonal pairs as named "double update", since they are affected by the rotations both from the same rows and columns. Numerical diagonal PEs perform rotation simultaneously in parallel with the updates of their respective columns and rows that are operated in off-diagonal PEs. Values are transmitted along their dataflow paths once their calculations are completed. The general dataflow is demonstrated in Fig. 2 as arrows, in which

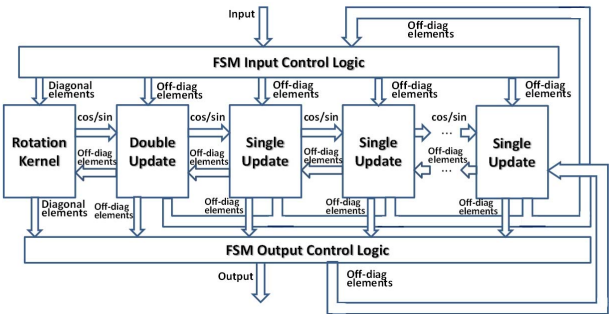

Fig. 3: Block diagram of the general 1D systolic array architecture for EVD.

the arrows labeled with ① and ② indicate the transmission of guest diagonal element and rotation angle parameter (*cos, sin*) respectively while the movements of host off-diagonal elements and guest off-diagonal elements are represented by the arrows labeled with ③ and ④ respectively. The host off-diagonal elements $x_{r+1,1}$, which move leftwards for rotations iteratively, continue to be updated while moving downward after being zeroed and then loop back to the end of the row when they have reached the bottom row of the PEs. The movement of guest off-diagonal elements $x_{r+1,r}$ follows their respective guest diagonal element $x_{r+1,r+1}$; guest off-diagonal elements loop back to the end of row when they are needed by subsequent updates.

To fit our design on a single chip, pipelined computational cores provide the opportunity to reuse the PEs with parallel calculations. One Jacobi Rotation PE is devised to perform all the Jacobi Rotations in a pipeline, while a series of pipelined off-diagonal single update PEs and one pipelined off-diagonal double update PE are used to simultaneously update groups of affected off-diagonal sub-matrices. Consequently, the architecture is converted into a one-dimensional systolic array as shown in Fig. 3 with the number of off-diagonal Single Update components determined by the dimension of the matrices and the resource capacity of the hardware.

A. Diagonal Jacobi Rotation Component

To zero out an off-diagonal element, Jacobi rotation is performed with its respective two diagonal elements in the same row or column. Jacobi Rotation can be performed through a series of addition, subtraction, multiplication, division and square root operations. Through expanding the rotation formulas, the rotation process is shown in Eqns. (3, 4), in which a_{pp}, a_{qq}, and a_{pq} represent two diagonal elements and an off-diagonal element respectively. Then, the process is optimized by shortening the latency and parallelizing the calculations. To balance resource allocation between the Jacobi rotation component and all the updating modules, whose computational latency increases linearly with the growth of matrix dimension, pipelined floating-point cores are shared by the calculations.

$$cos = \sqrt{\frac{(a_{qq}-a_{pp})^2+2*a_{pq}^2+|a_{qq}-a_{pp}|*\sqrt{(a_{qq}-a_{pp})^2+4*a_{pq}^2}}{(a_{qq}-a_{pp})^2+4*a_{pq}^2+|a_{qq}-a_{pp}|*\sqrt{(a_{qq}-a_{pp})^2+4*a_{pq}^2}}} \quad (3)$$

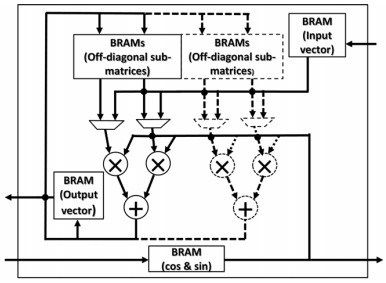

Fig. 4: Update component architecture.

$$sin = (sign)\sqrt{\frac{2*a_{pq}^2}{(a_{qq}-a_{pp})^2+4*a_{pq}^2+|a_{qq}-a_{pp}|*\sqrt{(a_{qq}-a_{pp})^2+4*a_{pq}^2}}} \quad (4)$$

B. Off-diagonal Single Update Component

$$Offdiag'_{host} = Offdiag_{host} \times cos - Offdiag_{guest} \times sin \quad (5)$$

$$Offdiag'_{guest} = Offdiag_{host} \times sin + Offdiag_{guest} \times cos \quad (6)$$

The off-diagonal single update component is responsible for updating the off-diagonal elements, which are affected by one Jacobi Rotation each iteration. Although the updating process consists of simple multiplications and addition or subtraction as is shown in eq. 5 and eq. 6, it is infeasible to fit an arbitrary number of updating components on a single chip. In our design, floating-point computational cores are employed to process the updates of sub-matrices of host off-diagonal elements and guest off-diagonal elements in a pipeline, in which limited number of PEs can perform large-scale updates in parallel. Every time the sub-matrices are completed with their update, a vector of host off-diagonal elements will be sent leftwards to the next off-diagonal single update component, and a vector of guest off-diagonal elements will be transmitted to external memory or looped back as input of another off-diagonal single update component according to the request.The architecture of our off-diagonal single update component is shown by the solid lines in Fig. 4.

C. Off-diagonal Double Update Component

The off-diagonal double update component is responsible to update sub-matrices of off-diagonals, which have to be processed with two updates successively each time, since both of their respective columns and rows are involving with the rotations. To integrate all of the updates affected by two rotations into one component, local memories are used to hold four triangular sub-matrices of off-diagonal elements, and the number of computational cores is at least twice as many used in the off-diagonal single update component in order to synchronize with the other components. An example off-diagonal double update component architecture is shown in Fig. 4 with both solid and dashed lines.

VI. Experiments and Evaluations

A. Implementation and Experimental Setup

To evaluate our design, we programmed our architecture on a single Xilinx Virtex-5 XC5VLX330 FPGA of the Convey HC-2 system [9]. In our implementation, we generated IEEE-754 double-precision floating-point calculators using the Xilinx Logic IP core generator. In the diagonal Jacobi Rotation component, eight rotations can be initiated for every 64 clock cycles with double-precision calculators as one divider, one square root, two adders and three multipliers, among which adders and multipliers were configured to use dedicated multiplier circuitry (DSPs). In each off-diagonal single update component, IP core generated Block RAMs are used to hold the sub-matrices of off-diagonal elements and rotation angle parameters, while one double-precision floating-point adder and two double-precision floating-point multipliers were implemented by dedicated multiplier circuitry (DSPs) and logics respectively.

B. Performance Analysis

In our design, maximally, 32 off-diagonal updating components can be allocated on our target FPGA, in which 31 off-diagonal single update components and one off-diagonal double update component are included. By evaluating our design at the frequency of 100 Mhz with 6 iterations, which was believed sufficient for convergence on matrices with certain thresholds, the performance of our design has demonstrated dimensional-dependent speedups from $1.5\times$ to $15.45\times$ for moderate- to large-sized matrix compared to optimized Matlab 7.10.0 software SVD solution that was processed on a 2.2 GHz dual core Intel Xeon processor with 16 GB installed memory as shown in Table I.

Fig. 5 demonstrates the quantitative comparison among dimensional dependent execution times of EVD processing by using different approaches. The blue line demonstrates the EVD performance by using our architecture while the performance of the Matlab 7.10.0 EVD routine running on the Intel platform is shown by the red line. The execution time of EVD/SVD solutions with Intel MLK 10.0.4 and NVIDIA 8800 GPU [2], both of which are using a 2.66 GHz Intel Core 2 Duo CPU, are depicted as green and purple lines respectively. By analyzing those data points in Fig. 5, although GPU-based solution has demonstrated better efficiency when matrix size grows over thousands, our design is more efficient for processing matrices up to 2000×2000.

Practically, the system performance of our design is dominated by the time consumption of rotations for small-scale applications; however, when the matrix size grows over a comparably large value such as 512, updates consume more time than rotating, which incurs a performance degradation. Additionally, to the best of our knowledge, [8] was the latest and only scalable architecture for FPGA-based EVD/SVD design; however, its performance suffered from the iterative design and low-capacity platform they employed, and these previous published results are slower than our results by two orders of magnitude with a matrix size limitation of 32×128.

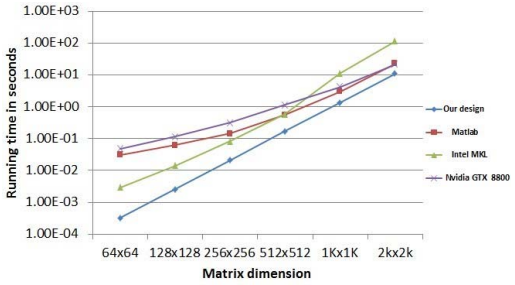

Fig. 5: EVD/SVD computation time (in seconds) for symmetric matrix by our design, Intel MKL and GPU.

TABLE I: Proposed Architecture performance in speed.

Dimension	Our Architecture	Matlab	Speedup
64×64	0.00202s	0.0312s	$15.45\times$
128×128	0.0091s	0.0624s	$6.4\times$
256×256	0.0320s	0.1428s	$4.46\times$
512×512	0.2558s	0.5446s	$2.2\times$
1024×1024	2.0290s	3.0607s	$1.5\times$

VII. Conclusion

An efficient reconfigurable FPGA-based hardware architecture is proposed to perform eigenvalue decomposition; which employed a novel modified Partitioned-Jacobi algorithm and a pipelined one dimensional systolic array. The analysis of our architecture demonstrates the scalability and dimensional dependent efficiency of our design.

Acknowledgements

This work is supported in part by the National Science Foundation (NSF) under awards CNS-1116810 and CCF-1149539.

References

[1] G. Golub and W. Kahan, "Calculating the Singular Values and Pseudo-Inverse of a Matrix," *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965.

[2] S. Lahabar and P. Narayanan, "Singular value decomposition on GPU using CUDA," in *Proceedings of IEEE International Symposium on Parallel Distributed Processing*, May 2009, pp. 1 –10.

[3] C. Kotas and J. Barhen, "Singular value decomposition utilizing parallel algorithms on graphical processors," in *Proceedings of OCEANS 2011*, Sept. 2011, pp. 1 –7.

[4] F. T. Brent, Richard P. Luk and C. V. Loan, "Computation of the singular value decomposition using mesh-connected processors," *Journal of VLSI Computer Systems*, pp. 243–270, 1985.

[5] M. Hestenes, "Inversion of matrices by biorthogonalization and related results," *Journal of the Society for Industrial and Applied Mathematics*, vol. 6, no. 1, pp. 51–90, 1958.

[6] R. P. Brent and F. T. Luk, "A systolic architecture for the singular value decomposition," Ithaca, NY, USA, Tech. Rep., 1982.

[7] A. Ahmedsaid, A. Amira, and A. Bouridane, "Improved SVD systolic array and implementation on FPGA," in *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT)*, Dec. 2003, pp. 35 – 42.

[8] L. Ledesma-Carrillo, E. Cabal-Yepez, R. de J Romero-Troncoso, A. Garcia-Perez, R. Osornio-Rios, and T. Carozzi, "Reconfigurable FPGA-Based unit for singular value decomposition of large m x n matrices," in *Proceedings of International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Nov.-Dec. 2011, pp. 345 –350.

[9] "The convey hc-2 computer architecture overview." [Online]. Available: http://www.conveycomputer.com/