Scalable Parallel Architecture for Singular Value Decomposition of Large Matrices

Unai Martinez-Corral*, Koldo Basterretxea†, and Raul Finker‡

*†Grupo de Diseño en Electrónica Digital (GDED) ‡Dept. Electronics Technology
University of the Basque Country (UPV/EHU)
Bilbao, Basque Country, Spain
*umartinez012@ikasle.ehu.es †koldo.basterretxea@ehu.es ‡raul.finker@ehu.es

Abstract—Singular Value Decomposition (SVD) is a key linear algebraic operation in many scientific and engineering applications, many of them involving high dimensionality datasets and real-time response. In this paper we describe a scalable parallel processing architecture for accelerating the SVD of large $m \times n$ matrices. Based on a linear array of simple processing units (PUs), the proposed architecture follows a double data-flow paradigm (FIFO memories and a shared-bus) for optimizing the time spent in data transfers. The PUs, which perform elemental column-pair evaluations and rotations, have been designed for an efficient utilization of available FPGA resources and to achieve maximum algorithm speed-ups. The architecture is fully scalable from a two-PU scheme to an arrangement with as many as $n/2$ PUs. This allows for a trade-off between occupied area and processing acceleration in the final implementation, and permits the SVD processor to be implemented both on low-cost and high-end FPGAs. The system has been prototyped on Spartan-6 and Kintex-7 devices for performance comparison.

Index Terms—Singular Value Decomposition, scalable architecture, adaptive threshold, CORDIC, co-processor, FPGA

I. INTRODUCTION AND RELATED WORK

The Singular Value Decomposition (SVD) of a $m \times n$ matrix $A$ is defined by:

$$U_{m \times m} \Sigma_{diag(n)} V_{n \times n}^T = A_{m \times n}$$

(1)

where $U$ and $V$ matrices are orthogonal, and $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_n)$ where $\sigma_1, \sigma_2, ..., \sigma_n$ are the singular values of $A$. Among the existing factorization algorithms for extracting quantitative information from (very) high dimensionality datasets, SVD is, on balance, one of the most accurate and numerically stable method. SVD is commonly used in the solution of unconstrained linear least square problems, the one-sided Jacobi variant (Hestenes-Jacobi algorithm) avoids data-dependence issues: an orthogonalization of pairs of rows/columns, data sharing is nearly non-conflicting, thus providing an opportunity for parallel processing. Brent, Luk and Van Loan’s [1] idea of an expandable square systolic array of simple $2 \times 2$ Processing Units (PUs) —named BLV— together with Sibul and Fogelsanger’s [2] proposal of using CORDIC Computer (CORDIC) for SVD was merged by Cavallaro and Luk [3] and a milestone was set. In fact, most of the subsequent parallel architectures for SVD proposed into the literature are based on this scheme. A linear array of $n/2$ processors was also proposed by the same authors, and Schreiber [4] went into this scheme in depth regarding undersized linear architectures, i.e. $pu < n/2$.

Since each PU tackles two rows and two columns in a two-sided Jacobi algorithm, focusing on embedded systems and large matrices, managing so much data can be a complex task. Besides that, any parallelisation attempt implies at least two PUs taking eight common elements. Given an $m \times n$ non-symmetric matrix, the one-sided Jacobi variant (Hestenes-Jacobi algorithm) avoids data-dependence issues: an orthogonal matrix $W$ is generated as a product of plane rotations, $Q_{ij}$, to matrix $A$, and exactly the same rotations are applied to $I_{n \times n}$ in order to get $V$. By applying the formula of Rutishauser [5], (2), in each rotation the angle, $\theta$, is chosen in such a way that the resulting column-pairs are orthogonal.

$$\theta = \frac{atan\left(\frac{2 \cdot (W_j * W_j)}{|W_j|^2 - |W_i|^2}\right)}{2}$$

(2)

The decomposition is then completed by getting the singular values of $A$, i.e. $\Sigma$ in (1), out of $W$, as these equal the $l_2$ norm or Euclidean length of its columns, $\sigma_i = ||W(:, i)||_2$.

This work was supported by the Spanish Ministry of Economy and Competitiveness, and European FEDER funds (grant TEC2010-15388), and by the Basque Government (grants IT733-13, S-PC12UN016, and S-PC13UN034).
Finally, $U$ is obtained by normalizing each column with its singular value, $U(:,i) = W(:,i)/\sigma_i$. Linear arrays to compute one-sided Jacobi have also been reported (see [6], [7], for instance).

Since FPGA-based processor implementation allows for the use of advanced design methods such as pipelining, parallelization and HW/SW co-design to achieve higher processing performance per unit area and power consumption, the design of specifically tailored SVD processing architectures has been a common approach in contemporary literature. In [8] the broadcasting of BLV is improved, and is claimed to be the first FPGA implementation of a SVD unit. In [9] a module compensation relaxation scheme is proposed to achieve shorter computation times at the expense of prolonging the time needed to converge. In [10] direct broadcasting to all off-diagonal PUs is implemented. While the mentioned proposals are based on two-sided Jacobi, in [11] Hestenes-Jacobi is used in a system made upon a PC and a FPGA-board plugged in a PCI slot, where dynamic reconfiguration is used to implement two configurations in a $4 \times 4$ processor scheme. In [12] the same PU design is synthesised along with the entire system in a FPGA, enhancing CORDIC instead of the PC and avoiding dynamic reconfiguration. In [13], a $4 \times 4$ complex SVD processor is proposed, which is composed of a vector product computation unit and eight customized CORDIC cores in a $2 \times 4$ scheme with a shared bus for each line.

In brief, since many published architectures attempt to theoretically reach full parallelization, mostly small to medium-size examples are described (matrix sizes up to $8 \times 8$, $40 \times 40$, $127 \times 32$ and $150 \times 150$) and, so far as the authors are aware, no practical optimal performance designs have been published considering parallel-computation of large matrices in FPGAs. Moreover, all mentioned works focus on the orthogonalization of a given matrix $A$, and neither data-transfer issues nor the computation of the auxiliary matrix $V$ are considered when analysing time and resource requirements. When addressed, it has been usually solved by doubling the resource usage [1], [13]. However, since effective computation-time has been heavily reduced in the past decades making the best of CORDIC and DSP blocks, the relative impact of data transfer times has become critical. In this sense, mimicking computer network schemes with limited communication resources leads to considering the non-simultaneity of data-transfers and computation. In consequence, a means for improving the SVD processing speed is customizing the Jacobi-like algorithms to achieve data-flow schemes that effectively map to reconfigurable devices to make the most of the available HW resources.

In this work, we present a scalable parallel processing architecture for a modified Hestenes-Jacobi SVD based on a linear array of PUs with a double data-flow paradigm (FIFO memories and a shared bus) for efficient data transfer. The design has been specifically tailored to the factorization of large matrices, but can be adapted to process square and nonsquare matrices of any size. Some internal arithmetic operations have been carefully designed, enhancing both CORDIC as in [3] and DSPs in MACC mode as in [11], to achieve a computing scheme that makes the most of embedded resources in modern FPGAs. At the same time, data-dependency has been thoroughly analysed to achieve maximum overlapping of stages in each PU.

II. A MODIFIED HESTENES-JACOBI ALGORITHM

Being iterative, the Hestenes-Jacobi algorithm goes through all column-pairs at least once in a minimum-length sequence named 'sweep'. A threshold value is needed to decide whether the orthogonalization process is finished. By setting this threshold, the user can obtain a trade-off between desired accuracy and computing-effort (processing time). A modification of the Hestenes-Jacobi algorithm named Active Adaptive Rutishauser (AARH) is proposed by the authors in [14]. The modified algorithm relies on the evaluation of the shifted minimum norm and the angle of Rutishauser (9). On top of that, columns are swapped before orthogonalization when $||W_i||_2 < ||W_j||_2$. This proposal outperforms previously published approaches in terms of the total number of sweeps/rotations required to achieve the same accuracy.

Considering the restrictions imposed by memory bandwidth when dealing with big matrices in embedded applications, moving data only when needed and performing computations in parallel with column transfers can help accelerating the SVD. Cyclic sequences seem to be the most suitable when seeking minimum data transfer, since a column $i$ may be kept in a PU while computing $n-i$ pairs, that is from $j = i + 1$ to $j = n - 1$. Besides that, when any $||W_i||_2$ or $||W_j||_2$ is null no orthogonalization is required, and due to the implicit sorting in AARH, the rank of the matrix can be estimated online, rather than waiting until convergence is met. In consequence, the computation can be notably shortened after each sweep. The flowchart of the modified algorithm is shown in Figure 1, where $ns$ tracks the smallest index with a null norm during a

![Fig. 1. Active Adaptive Rutishauser (AARH).](image)
sweep, and $nl$ gets the rank at the end of it. In the beginning of the next sweep a problem of size $m \times nl$ is computed. If a full-rank matrix is processed, at the end $ns = nl = n - 1$.

### III. A data driven computing scheme

The proposed architecture for the parallelization of the algorithm described in the previous section is based on the interconnection of $pu$ basic processing units or PUs. As a result of the adaptability of AARH, not all column-pair computations take the same time, so tight schedules for pure parallel-working PUs would not be optimal. We designed a "loose" linear systolic array architecture by fitting one FIFO parallel-working PUs would not be optimal. We designed a "loose" linear systolic array architecture by fitting one FIFO

**Fig. 2.** Proposed processing architecture for $pu = 3$: three blocks of main memory and a bus with three channels (left); Processing Unit (PU) core design (center) and column-pair schedule for $n = 8$ (right).

Since minimum convergence time is achieved when $(i, j)$ pairs are computed before any $(i + k_i, j + k_j)$, it is sensible to assign sequential $i$ indexes to the PUs placed in a linear scheme and make the $W_j$ columns cross the array at most $\beta = 1 + (nl\mod pu)$ times in a sweep. This is exposed in Figure 2 (right) for $pu = 3$ and $nl = 8$. Since each $W_j$ column is sent directly from the PU computing column-pair $(i, j)$ to the PU waiting to start $(i + 1, j)$, the number of required transfers from/to the main memory is reduced to $\gamma_{nl}^V = \beta \cdot (pu \cdot (\beta - 1) + 2 \cdot (nl \mod pu))$, which results in saving about 65% of the required transfer time: $\gamma_{F}^{V} / \gamma_{F}^{0} = 42.85\%$, $\gamma_{V}^{50} / \gamma_{V}^{0} = 34.69\%$, $\gamma_{P}^{50} / \gamma_{P}^{0} = 33.5\%$. This is achieved with no penalty in data dependence, i.e. convergence, allowing for the simplest architecture with a unique data-flow: from the reading port of the main memory, through the array of PUs and back to the writing port (see Figure 2, left).

Furthermore, since columns in $V_i$ and $V_j$ are only to be computed if the corresponding $W_i$ and $W_j$ have been rotated, a secondary data-flow is adopted to move data without crossing the whole array. This has been accomplished by implementing two unidirectional shared buses to produce a disjoint full-duplex communication channel, which involves a multiplexor-based crossbar-switch and an arbiter for managing the connections. This secondary data-flow is used to push $W_i$ columns into the main memory once all the corresponding computations are done, as well as to pull/push $V_i$ (which are transferred just once per sweep) and $V_j$ columns (each time needed). The maximum number of data-transfer cycles required per sweep for the orthogonalization of $W$ and $V$ is $n \cdot \gamma_{nl}^V + m \cdot \gamma_{nl}^P$, although it can be expected to be less since many transfers may be overlapped (e.g. if $PU_p$ is requesting access to push $W_j$ and $PU_{p+1}$ is requesting to pull it, the main memory may be bypassed). If the resources in the target FPGA suffice, and in the case that the main memory is split into different blocks, multichannel ($ch > 1$) buses can be implemented to reduce potential access collisions. Ideally, no collision will exist if $n$ memory blocks ($mem$) are used, holding $m + n$ pieces each (a
column of \( W \) and its corresponding of \( V \), and \( pu+1 \) channels are implemented.

### B. The Processing Unit (PU)

As exposed in Figure 1, the Hestenes-Jacobi algorithm can be analysed in two separate non overlapping stages: 1) orthogonalization and 2) factorization. During orthogonalization PUs are required to check whether \( \| W_i \|_2^2 < \| W_j \|_2^2 \) and evaluate each pair of columns according to AARH, which involves computing (2). If the columns in a pair are not orthogonal, a Givens’ rotation has to be performed. Arithmetic operations to be performed by the PUs have been carefully chosen to achieve a hardware-friendly scheme. Fixed-point arithmetic has been used, all the fixed shifts have been hardwired to avoid barrel-shifters, and unrolled pipelined realizations have been selected to improve throughput. Embedded DSP blocks have been directly used in MACC mode to compute the square Euclidean norms and vector multiplications \( \| W_i \|_2^2, \| W_j \|_2^2 \), and \( W_i \ast W_j \). CORDIC is used in vectoring/evaluation/accumulation mode to compute the angle of Rutishauser (2) and it has been directly compared against the threshold as described in [14]. If rotation is required, the same CORDIC core has been used in rotating mode to effectively perform Givens’ rotations, thus all the computations have been reduced to addsubs or fixed shifts, except for the computation of the square Euclidean norms and vector products.

As exposed previously, the SVD may be accelerated by computing in parallel with data transference. Since fine-grained pipelining has nearly no hardware cost in modern FPGAs, transference can be executed at maximum rate —one piece per clock cycle— and high throughput may be achieved when managing large datasets. Figure 2 (center) shows a simplified scheme of the nuclear PU design, which is composed of three modules (EVALUATION, CORDIC and CACHE), and two cooperative FSMs to manage an internal multiplexor-based crossbar-switch to share data between the modules.

### C. Results

The proposed architecture has been implemented in both Spartan-6 and Kintex-7 devices by Xilinx. Resource utilization and system specifications for the orthogonalization, i.e. to obtain \( W \) and \( V \), are exposed in Table I. Processing times have been measured considering \( A_{m \times n} \) and \( I_{n \times n} \) are already loaded into the main memory.

### IV. CONCLUSION AND FUTURE WORK

A parallel processing scheme based on a scalable linear array of processing units has been developed, which adopts a double data-flow paradigm to optimize data transfers by avoiding unnecessary transmissions and computations. Speedup-factors up to \( 600 \times \) compared to a softcore microprocessor-based solution (Microblaze) have been obtained, validating previously fixed step simulation-based speed-up and excellent scalability figures [14]. We have not found any previously proposed architectures to compare with in terms of resource utilization figures, either because small matrices are used or because insufficient details are reported.

### Regarding future work, redundant arithmetic-based CORDIC designs, the use of square root and division free Givens’ rotations, and designing ad-hoc estimators to compute Euclidean norms and the \( \text{atan} \) function are some of the means for upgrading we would like to explore. For improved area efficiency, two designs can be explored, customizing the internal connections for each function, and using dynamic partial reconfiguration to switch from orthogonalization to factorization once convergence is met.

### REFERENCES


