Application of On-Line Arithmetic Algorithms to the SVD Computation: Preliminary Results*

Paul K.-G. Tu

Advanced Workstation Division
IBM Corporation
Austin, TX 78758

Miloš D. Ercegovac

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024

Abstract

A scheme for the singular value decomposition (SVD) problem, based on on-line arithmetic, is discussed. The design, using radix-2 floating-point on-line operations, implemented in the LSI[1] *HCMOS gate array technology, is compared with a compatible conventional arithmetic implementation. The preliminary results indicate that the proposed on-line approach achieves a speedup of 2.4-3.2 with respect to the conventional solutions, with 1.3 - 5.5 more gates and more than 6 times fewer interconnections.*

1 Introduction

In this paper we explore the effectiveness of implementing numerical computations with on-line arithmetic algorithms. Of interest are the characteristics of computations that make them suitable for on-line implementation and the cost and performance features of such on-line computations. Since on-line algorithms operate in a digit-pipelined fashion, it is essential that the critical path of its application contain long sequences of arithmetic operations. The singular value decomposition (SVD)[13] is one such application. Other approaches in implementing SVD are described in [2, 4, 6, 10, 8, 7].

In order to conduct comparison with conventional approaches, we give estimates of the complexity and performance parameters of the on-line approach based on the LSI gate array designs given in [15, 16, 17]. The on-line algorithms have been implemented according to the LSI Logic gate array design guidelines using Viewlogic Workview CAD tools[18]. Complexity figures are calculated directly from the designs, and delays are measured from library components using the data given in [15]. Although the discussed on-line SVD system has not been completely implemented, the presented gate counts and delays provide, in our opinion, useful estimate about the feasibility and performance of on-line approach. The results presented here are preliminary and limited. We did not study in depth the effects of cancellation of leading digits in floating-point additions which present challenging design problems. We do, however, discuss the problems and outline some solutions.

*Supported in part by the NSF grant No. MIP-8813340.

[1]LSI is a trademark of LSI Logic Corporation.

We have not compared the cost/performance of the on-line SVD with other special-purpose designs [6, 2] because of incompatible technology or lack of design details.

In the following, we first give an overview of the SVD algorithm. Then the on-line scheme for the SVD computation is presented. It is assumed that standardized single-operation on-line units are used for the implementation. The on-line scheme is compared with the conventional approach in Section 4.

2 The SVD algorithm

The singular value decomposition (SVD) is one of the basic matrix operations required for many important modern signal processing computations[13]. The SVD of a matrix $\mathbf{A}$ is defined as the factorization

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$$

where $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, and Σ is a diagonal matrix with non-negative diagonal elements. For many real-time signal processing applications, numerically stable SVD algorithms that are suitable for implementation on parallel computer architectures are desirable.

We adopt the algorithm by Brent, Luk and van Loan[12] which computes the SVD of an $n \times n$ matrix $\mathbf{A}$ on a mesh-connected processor array. It has been shown in [11] that this algorithm converges when n is odd, and does not always converge when n is even. In our discussion on-line arithmetic is used as the low level realization of arithmetic operations of a numerical computation, and it is assumed that for the on-line computation to generate the correct result, the high level algorithm must be numerically correct when realized by conventional parallel arithmetic algorithms. Our emphasis is on the implementation efficiency and performance, not the convergence of the computation. For the sake of clarity in our illustration, we assume that $\mathbf{A}$ is real and n is even. The same conclusions apply when n is odd.

In this scheme, the array consists of $n/2 \times n/2$ processors, where each processor operates on a 2×2 submatrix of $\mathbf{A}$ (Fig. 1). A series of 2×2 SVDs is performed on submatrices along the main diagonal of $\mathbf{A}$, indicated by the shaded areas in Fig. 1. Each 2×2 SVD is realized

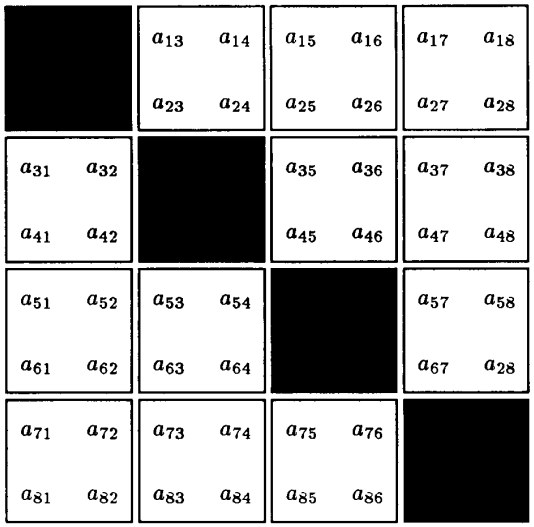

Figure 1: Mapping a matrix onto a processor array ($n = 8$)

by a 2-sided rotation that diagonalizes the submatrix. The rotation angles are computed by Algorithm FHSVD [12].

Algorithm FHSVD

$\mu_1 = a_{i+1,i+1} - a_{i,i} \quad \nu_1 = a_{i+1,i} + a_{i,i+1}$
if $|\nu_1| \leq \epsilon|\mu_1|$ **then**
$\chi_1 = 1 \quad \sigma_1 = 0$
else
$$\rho_1 = \frac{\mu_1}{\nu_1} \quad \tau_1 = \frac{\text{sign}(\rho_1)}{|\rho_1| + \sqrt{1+\rho_1^2}}$$
$$\chi_1 = \frac{1}{\sqrt{1+\tau_1^2}} \quad \sigma_1 = \chi_1\tau_1$$

$\mu_2 = a_{i+1,i+1} + a_{i,i} \quad \nu_2 = a_{i+1,i} - a_{i,i+1}$
if $|\nu_2| \leq \epsilon|\mu_2|$ **then**
$\chi_2 = 1 \quad \sigma_2 = 0$
else
$$\rho_2 = \frac{\mu_2}{\nu_2} \quad \tau_2 = \frac{\text{sign}(\rho_2)}{|\rho_2| + \sqrt{1+\rho_2^2}}$$
$$\chi_2 = \frac{1}{\sqrt{1+\tau_2^2}} \quad \sigma_2 = \chi_2\tau_2$$

$$c_i^L = \chi_1\chi_2 + \sigma_1\sigma_2 \quad s_i^L = \sigma_1\chi_2 - \chi_1\sigma_2$$
$$c_i^R = \chi_1\chi_2 - \sigma_1\sigma_2 \quad s_i^R = \sigma_1\chi_2 + \chi_1\sigma_2$$

end.

Then the rotation is computed as

$$\begin{bmatrix} a'_{i,i} & 0 \\ 0 & a'_{i+1,i+1} \end{bmatrix} = \begin{bmatrix} c_i^L & s_i^L \\ -s_i^L & c_i^L \end{bmatrix}^T \begin{bmatrix} a_{i,i} & a_{i,i+1} \\ a_{i+1,i} & a_{i+1,i+1} \end{bmatrix} \begin{bmatrix} c_i^R & s_i^R \\ -s_i^R & c_i^R \end{bmatrix}$$

where c_i^L, s_i^L, c_i^R and s_i^R are the cosine and sine values of the left and right rotation angles, respectively.

During each iteration, $n/2$ 2×2 SVDs are performed in parallel. Values of the left rotation angles are passed to processors of the corresponding rows while values of the right rotation angles are passed along the corresponding columns, where 2×2 rotations are performed on the submatrix in each processor,

$$\begin{bmatrix} a'_{i,j} & a'_{i,j+1} \\ a'_{i+1,j} & a'_{i+1,j+1} \end{bmatrix} = \begin{bmatrix} c_i^L & s_i^L \\ -s_i^L & c_i^L \end{bmatrix}^T \begin{bmatrix} a_{i,j} & a_{i,j+1} \\ a_{i+1,j} & a_{i+1,j+1} \end{bmatrix} \begin{bmatrix} c_j^R & s_j^R \\ -s_j^R & c_j^R \end{bmatrix}$$

The 2-sided 2×2 rotation is computed as

$$a'_{i,j} = c_j^R(c_i^L a_{i,j} - s_i^L a_{i+1,j}) - s_j^R(c_i^L a_{i,j+1} - s_i^L a_{i+1,j+1})$$
$$a'_{i,j+1} = s_j^R(c_i^L a_{i,j} - s_i^L a_{i+1,j}) + c_j^R(c_i^L a_{i,j+1} - s_i^L a_{i+1,j+1})$$
$$a'_{i+1,j} = c_j^R(s_i^L a_{i,j} + c_i^L a_{i+1,j}) - s_j^R(s_i^L a_{i,j+1} + c_i^L a_{i+1,j+1})$$
$$a'_{i+1,j+1} = s_j^R(s_i^L a_{i,j} + c_i^L a_{i+1,j}) + c_j^R(s_i^L a_{i,j+1} + c_i^L a_{i+1,j+1})$$

After each iteration, rows and columns of the matrix are exchanged between adjacent processors according to the *parallel ordering* which is described in [1]. Under this ordering, for each i and j, such that $1 \leq i < j \leq n$, columns i and j of $\mathbf{A}$ are paired and rotated in the same column of processors exactly once in every $n-1$ consecutive iterations which are referred to as a *sweep*. Likewise, for each $1 \leq i < j \leq n$, rows i and j of $\mathbf{A}$ are paired and rotated in the same row of processors exactly once in every sweep. To simplify control, a fixed number of S sweeps is performed to diagonalize $\mathbf{A}$. It is shown in [12] that $S < 9$ for $n \leq 50$.

Assuming that there is no data broadcasting among processors, Figure 2 shows the processor array by Brent, Luk and van Loan. The dashed arrow lines represent the transmission of the rotation angles and the solid arrow lines represent the transmission of matrix elements. Processors on the main diagonal compute the rotation angles and the main diagonal elements of the matrix, while those not on the main diagonal perform 2×2 rotations.

The performance bottleneck of the SVD is clearly the calculation of the rotation angle, which contains long sequences of sequential operations and cannot be effectively sped up by conventional techniques.

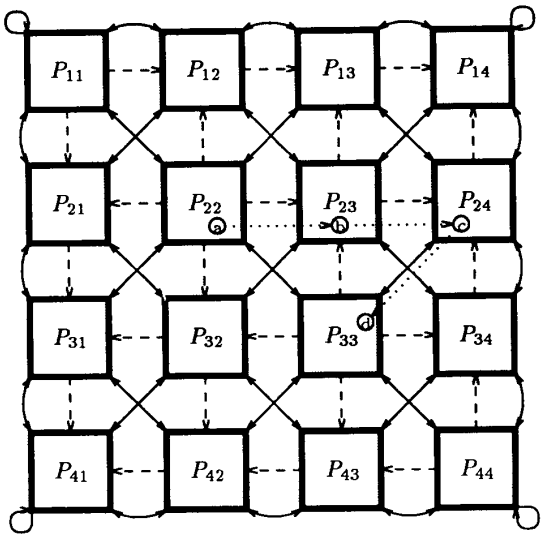

Figure 2: Systolic array for the SVD by Brent, Luk and van Loan ($n = 8$)

3 On-line scheme for computing the SVD

We now describe the SVD scheme based on on-line arithmetic[3, 5]. In this scheme each arithmetic operation in the computation is realized by a single-operation on-line unit. All on-line units have the same interfaces, so successive units can be interconnected directly with no intermediate data conversion. We assume that the inter-processor transmission of the angles and the matrix elements are on-line. On-line algorithms for floating-point operations used are given in the appendix. Their design appears in [15] and [17].

For each on-line operation, it is assumed that the input operands are quasi-normalized, which is defined as $|x| \in [\frac{1}{4}, 1)$. The output is also quasi-normalized. This is achieved by incorporating post-normalization step into the on-line algorithms for addition, multiplication and division operations. The on-line result of square root operation is always quasi-normalized. Details of post-normalization steps for different operations are given in the Appendix. The most-significant bit of the mantissa is always non-zero thus making the sign determination simple. This also holds in the case of cancellation of the most significant digits, since no output is delivered before the result is quasi-normalized. While the on-line delays of multiplication, division and square root given in Table 3 are bounded by 7, 7 and 4, respectively, the on-line delay of addition can be as large as 56 due to cancellation of the most significant digits. This is a serious problem that has not been included in the perfromance analysis in this paper. However, the design includes necessary on-line buffers to synchronize inputs arriving out of phase due to cancellation.

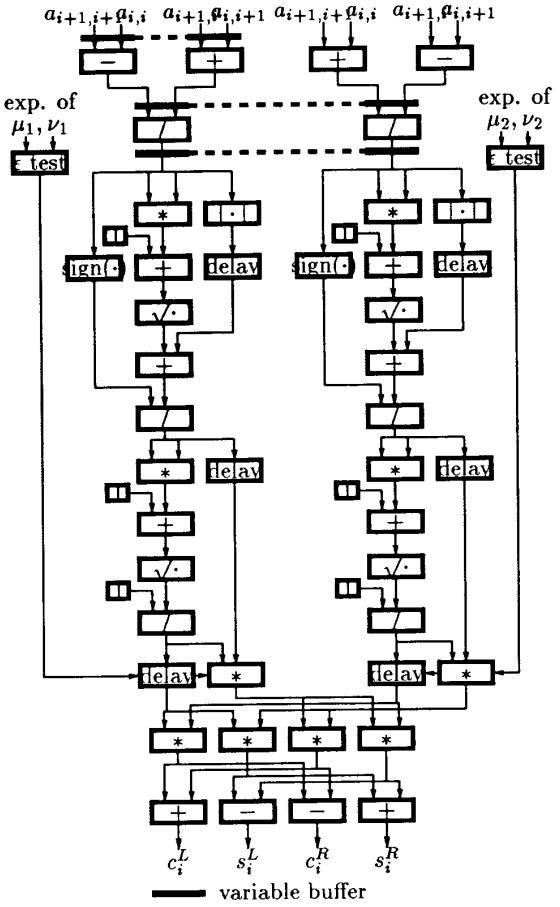

Figure 3: On-line scheme for algorithm FHSVD

3.1 Complexity of the on-line SVD

Figure 3 shows the on-line network for the FHSVD algorithm. The ϵ tests in the FHSVD algorithm are performed concurrently with the computation of ρ_i, τ_i, χ_i and σ_i. If a test succeeds, the corresponding variables are forced to 1 and 0, respectively. Two types of delay buffers are used in the implementation. Fixed delay buffers (FDB) are used to synchronize the arrival of operands for some of the on-line units over the paths with a fixed difference in on-line delays. The out-of-phase inputs, caused by cancellation, are synchronized through variable delay buffers (VDB). These buffers use a design similar to the mantissa alignment [15] and have an estimated cost of 1,150 gates/buffer. The sign of a number can be obtained directly from the sign bit of the first mantissa digit. Constants can be stored in registers and transmitted to the on-line units when needed. The control of on-line units is simple and it is not included in this study.

The on-line network for the rotation computation of

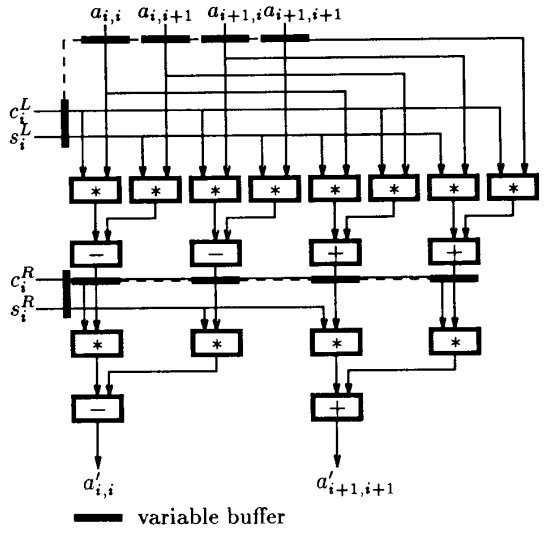

Figure 4: On-line scheme for rotation of a diagonal processor

main diagonal processors is shown in Figure 4. We assume that six inputs to the upper and lower level multipliers are synchronized using 6 variable delay buffers each. Figure 5 shows the on-line network for the rotation computation of off-diagonal processors. The number of each type of arithmetic units and estimates of their gate count for each processor are given in Table 1 for radix-2 computation, based on the designs described in [15]. The cost of postnormalization is included. Each input and output data requires 10 pins, 8 for the exponent and 2 for the mantissa. Let G_{diag} and G_{off} denote the gate count for each main diagonal and off diagonal processor, respectively. The total cost of an $n/2 \times n/2$ processor array, which computes the SVD of an $n \times n$ matrix, is

$$G_{\frac{n}{2}\times\frac{n}{2}} = G_{diag} \times \frac{n}{2} + G_{off} \times \left(\left(\frac{n}{2}\right)^2 - \frac{n}{2}\right) \quad (1)$$

3.2 Performance of the on-line SVD

The SVD computation is composed of the rotation angle calculation and the rotation calculation. The operations in the critical paths of these calculations are listed in Table 2. Let T_{cyc} denote the *iteration cycle time*, defined as the number of clock cycles between starts of consecutive iterations, and T_{ite} the *complete iteration time*, which is the number of clock cycles for a complete iteration. Assume that it takes one clock cycle for data to pass through a processor, and the exchange of matrix elements between processors takes no extra time. Then the *complete sweep time*, defined as the number of clock cycles to compute a sweep, which consists of $n-1$ iterations, is

$$T_{swe} = (n-2) \cdot T_{cyc} + T_{ite} \quad (2)$$

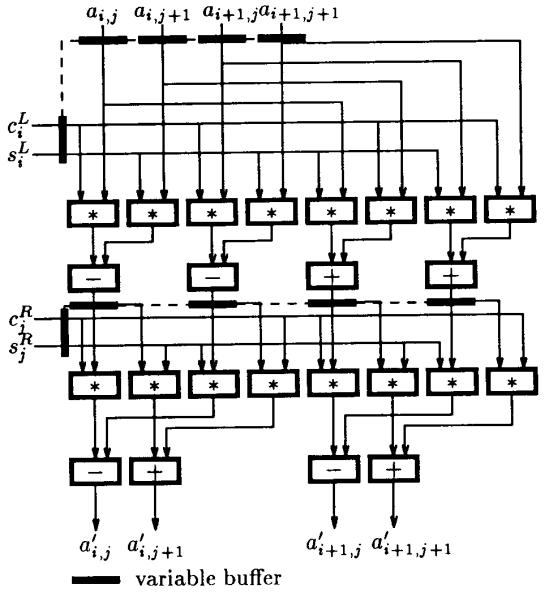

Figure 5: On-line scheme for rotation of an off-diagonal processor

Arith. unit	Gate count	Data pins	Main diag. proc. ang. units	rot. units	total units	cost	Off diag. proc. rot. units	cost
Add	1337	30	14	6	20	26700	8	10700
Mult	3941	30	10	12	22	86700	16	63100
Div	4261	30	6	0	6	25600	0	0
Sqrt	4261	20	4	0	4	17000	0	0
VD Buffer	1150		5	12	17	19950	12	13800
FD Buffer	100		6	0	6	600	0	0
ϵ Tests	200		2	0	2	400	0	0
Total cost			~ 176950				~ 87600	

Table 1: Processor cost of on-line SVD

Computation	+/−	×	÷	$\sqrt{}$
Angle calc.	5	4	3	2
Rotation	2	2		

Table 2: Critical path operations for the SVD

Operation	minimum step-time	δ_{min}	δ_{max}
Add*	14 t_g	5 *cycles*	5 *cycles*
Mult	15 t_g	5 *cycles*	7 *cycles*
Div	17 t_g	6 *cycles*	7 *cycles*
Sqrt	17 t_g	4 *cycles*	4 *cycles*

* no cancellation of leading digits

Table 3: On-line delay and step-time for radix-2 on-line units

The total number of clock cycles for the SVD consisting of S sweeps is

$$T_{svd}=(S\cdot(n-1)-1)\cdot T_{cyc}+T_{ite} \quad (3)$$

When post-normalization is performed in an on-line operation, it costs an extra cycle to the on-line delay since the second digit of the result is examined before determining whether the first digit can be output. When the result contains leading zeros, the zeros are discarded and more digits are examined. In the optimal case, there is no leading zero generated in the mantissa, while in the worst case (assuming no leading digit cancellation for addition), up to 1 (addition), 2 (division) or 3 (multiplication) leading zeros may be present. Based on the designs presented in [15], the required minimum step-time and minimum and maximum on-line delays of the radix-2 implementation of the basic operations are listed in Table 3. The step-time is measured in units of t_g, which is roughly the delay of a nand gate with no more than 4 inputs. In the following discussion, we define *cycles* as the maximum of the minimum required step-time for all individual operations. We also assume δ_{max} for each on-line unit. The delays of the components used in the designs are listed in [15]. The on-line delays for the rotation angle calculation and the rotation calculation, denoted as Δ_{ang} and Δ_{rot}, respectively, are

$$\Delta_{ang}=82\ cycles, \Delta_{rot}=24\ cycles$$

The iteration cycle time of the on-line SVD is limited by two factors. 1. The availability of input. In the SVD computation, the rotation angle calculation of each iteration requires the result of the rotation from the previous iteration. The time between the start of the rotation angle calculation and the arrival of data for the next iteration, denoted as Δ_{ite}, includes the on-line delays of the rotation angle calculation, of transmitting the computed angles through one processor, and of the rotation calculation on an off-diagonal processor, as is indicated in Figure 2 by the dotted path $a\rightarrow b\rightarrow c\rightarrow d$. For radix-2 computation, we have

$$\begin{aligned}\Delta_{ite}&=\Delta_{ang}+\delta_{pas}+\Delta_{rot}\\&=82+1+24=107\ cycles\end{aligned}$$

where δ_{pas} is the number of cycles for data to pass through a processor and is assumed to be 1.

2. The availability of the arithmetic units. For each on-line unit, the next iteration can not start until it has finished the computation of the previous iteration. Since in on-line computations all units work in pipelined fashion, the cycle time between iterations must be no less than the largest busy cycle time of all on-line units in the critical path of the computation. Assuming $m+1$ output digits are computed, the busy cycle of an on-line unit is

$$t_{busy}=\delta+m$$

where δ is the on-line delay of the on-line unit. For radix-2 computation, division has the largest busy cycle with $\delta=7$. For double precision computation of 8 exponent bits and 56 mantissa bits, we have

$$t_{busy}=63\ cycles$$

Both factors mentioned above present lower bounds on T_{cyc}. Assuming that the matrix size $n\geq 4$, we have

$$T_{ite}=\Delta_{ang}+(\frac{n}{2}-2)\cdot\delta_{pas}+\Delta_{rot}+m$$

then for radix-2 double precision floating-point computation, we have

$$\begin{aligned}T_{cyc}&=\Delta_{ite}=107\ cycles\\T_{ite}&=160+\frac{n}{2}\ cycles\\T_{swe}&=(107+\frac{1}{2})\cdot n-54\ cycles\\T_{svd}&=S\cdot(n-1)\cdot 107+\frac{n}{2}+53\ cycles\end{aligned}$$

The hardware utilization is measured in terms of *unit* $\cdot$ *cycle*, which is defined as an arithmetic unit resource for one clock cycle during the computation. The total amount of resource available for the SVD computation is

$$\text{total resource}=(\text{total number of units})\times T_{svd}$$

and the amount of resources utilized during the computation is the summation of busy cycles of all units

$$T_{busy}\equiv\sum_{\text{all units}} t_{busy}$$

The utilization factor of the computation is defined as the ratio of T_{busy} and the total resource,

$$U\equiv\frac{T_{busy}}{\text{total resource}}$$

For the on-line SVD computation, each main diagonal processor has 20 addition units, 22 multiplication units, 6 division units and 4 square root units. Each off-diagonal processor has 8 addition units and 16 multiplication units. Assuming the worst case on-line delay (δ_{max}) for each on-line operation, and no leading digit calcellation in addition, the busy cycle for each addition, multiplication, division and square root unit is 61,

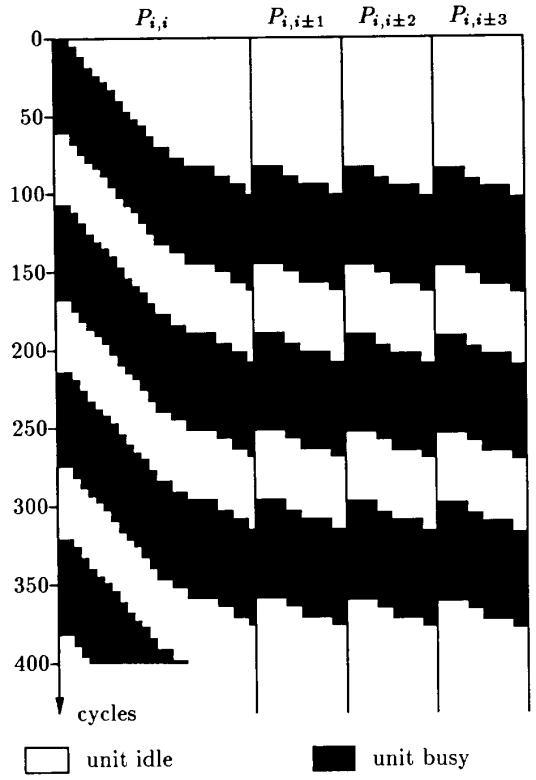

Figure 6: Timing diagram for on-line SVD ($n = 8$)

63, 63 and 60, respectively. The utilization factor for the on-line SVD is

$$U_{OL}=\frac{\left(3224\cdot\frac{n}{2}+1496\cdot\left(\left(\frac{n}{2}\right)^2-\frac{n}{2}\right)\right)\cdot(n-1)\cdot S}{\left(52\cdot\frac{n}{2}+24\cdot\left(\left(\frac{n}{2}\right)^2-\frac{n}{2}\right)\right)\cdot T_{svd}}$$

It can be shown that for $S \geq 10$ and $n \geq 8$, we have $U_{OL} > 0.57$. Figure 6 is a timing diagram that shows the busy and idle periods of each of the main diagonal and off-diagonal processors.

4 Comparisons

We now compare the on-line implementation with conventional special purpose designs for the SVD computation. The delays are measured in terms of *gate delays*, denoted as t_g, which is approximately the delay time of a nand gate with no more than 4 inputs. The effect of fanout is ignored in the comparison since it depends on low level design details, and we assume that it affects designs of both approaches similarly. Costs of the designs are measured in terms of LSI *equivalent gates*, which is defined in [9]. The data format of input and output of each arithmetic operation is double precision floating-point with 8-bit exponent and 56-bit mantissa.

Arith. unit	Main diag. proc. # inputs	Main diag. proc. num.	Off diag. proc. # inputs	Off diag. proc. num.
Add	5	8	2	8
Mult	3	12	2	16
	2	4		
Div	3	4		
Sqrt	2	4		
Total	2	8	2	24
	3	16		
	5	8		

Table 4: Multiplexers for arithmetic units (Case 1)

The designs of the on-line and conventional implementations are based on components available from the LSI HCMOS gate array libraries[9]. The cost and performance figures of the components for both approaches are based on the discussion in [15]. Conventional floating-point algorithms used in the study correspond to typical designs described in the literature or implemented on commercial chips. The floating-point adder uses a barrel shifter and a 2-level carry-lookahead mantissa adder, performing a double-precision floating-point addition in two clock cycles. The floating-point multiplier uses radix-4 recoded multiplier, a tree of carry-save adders, and a 2-level carry-lookahead adder to produce a double- precision product in six clock cycles. The floating-point radix-4 divider corresponds to Taylor's scheme[14] augmented with the on-the-fly quotient converter. It uses redundant remainders. Double-precision quotient is obtained in 30 cycles. Similar characteristics are assumed for the floating-point square-root unit.

From Table 3, the step-time of the on-line implementation is 16 t_g. The step-time of the conventional approach is assumed to be that of the 56-bit parallel adder, which is 21 t_g. The difference of step-time is taken into consideration when comparing the schemes.

Case 1: Maximum parallelism

In this case we assume for the conventional approach that there are as many arithmetic units available as needed by the computation whenever data is ready. For the processors on the main diagonal of the array, there are 4 adders, 8 multipliers, 2 dividers and 2 square root units. During each iteration, each adder is used 5 times, each multiplier 3 times, each divider 3 times and each square root unit 2 times. For the off-diagonal processors, there are 4 adders and 8 multipliers. Each adder and multiplier is used 2 times per iteration. Table 4 lists the multiplexers that are needed for each processor. The total cost of each main diagonal processor is

$$\begin{aligned}G_{diag}=&4\cdot g_{add}+8\cdot g_{mult}+2\cdot g_{div}+2\cdot g_{sqrt}\\&+8\cdot g_{mux2}+16\cdot g_{mux3}+8\cdot g_{mux5}\\\approx&149,400\end{aligned}$$

Arith. unit	Main diag. proc. # inputs	num.	Off diag. proc. # inputs	num.
Add	5	8	4	4
Mult	3	12	4	8
	2	4		
Div	3	4		
Sqrt	2	4		
Total	2	8	4	12
	3	16		
	5	8		

Table 5: Multiplexers for arithmetic units (Case 2)

and the cost of each off diagonal processor is

$$G_{off}=4\cdot g_{add}+8\cdot g_{mult}+24\cdot g_{mux2}$$
$$\approx 120,600$$

The number of cycles for each arithmetic operation are

$$t_{add}=2\ cycles$$
$$t_{mult}=6\ cycles$$
$$t_{div}=30\ cycles$$
$$t_{sqrt}=29\ cycles$$

From the critical path operations given in Table 2, the number of cycles for the rotation angle calculation and the rotation calculation are

$$T_{ang}=5\cdot t_{add}+4\cdot t_{mult}+3\cdot t_{div}+2\cdot t_{sqrt}=182$$
$$T_{rot}=2\cdot t_{add}+2\cdot t_{mult}=16$$

Then we have the iteration cycle time and the complete iteration time for $n\geq 4$,

$$T_{cyc}=T_{ang}+\delta_{pas}+T_{rot}=199\ cycles$$
$$T_{ite}=T_{ang}+(\frac{n}{2}-2)\cdot\delta_{pas}+T_{rot}$$
$$=196+\frac{n}{2}\ cycles$$

Case 2: Reduced complexity

Assume that each conventional processor on the main diagonal of the array still has 4 adders, 8 multipliers, 2 dividers and 2 square root units, but let the off diagonal processors have 2 adders and 4 multipliers each. Each off diagonal processor performs the rotation computation twice per iteration, each time computing the rotations of 2 of the matrix elements associated with the processor. If a matrix element is going to be transmitted to a main diagonal processor, it is always computed in the first run. Each adder and multiplier of an off diagonal processor is used 4 times per iteration. Table 5 shows the number of multiplexers used in each processor. The cost for each off diagonal processor is

$$G_{\text{off}}=2\cdot g_{add}+4\cdot g_{mult}+12\cdot g_{mux4}$$
$$\approx 62800$$

Because of the long computation delay of the rotation angle calculation performed by the main diagonal processors, T_{cyc} remains the same as in case 1. The complete iteration time becomes

$$T_{ite}=T_{ang}+(\frac{n}{2}-2)\cdot\delta_{pas}+2\cdot T_{rot}$$
$$=212+\frac{n}{2}\ cycles$$

The utilization factor for this scheme is

$$U_{C2}=\frac{\left(468\cdot\frac{n}{2}+112\cdot\left(\left(\frac{n}{2}\right)^2-\frac{n}{2}\right)\right)\cdot(n-1)\cdot S}{\left(16\cdot\frac{n}{2}+6\cdot\left(\left(\frac{n}{2}\right)^2-\frac{n}{2}\right)\right)\cdot T_{svd}}$$

and for $n=8$, $S=10$ we get $U_{C2}=0.124$.

Case 3: Maximum utilization

Assume that each main diagonal processor has one each of adder, multiplier, divider and square root unit, and each off-diagonal processor has one adder and one multiplier. For each iteration, each diagonal processor performs 20 additions, 22 multiplications, 6 divisions, 4 square root operations, and each off diagonal processor performs 8 additions and 16 multiplications. Each processor needs about 12 registers for buffering input, output and intermediate results. The complexity of the processors are

$$G_{diag}=g_{add}+g_{mult}+g_{div}+g_{sqrt}\approx 26800$$
$$G_{off}=g_{add}+g_{mult}\approx 16800$$

Assume that the order of performing the arithmetic operations is such that those operations in the critical path always proceed when ready. The performance figures are

$$T_{ang}=235\ cycles$$
$$\Delta_{ang}=221\ cycles$$
$$T_{rot}=\begin{cases}74\ cycles & \text{diagonal processor}\\ 98\ cycles & \text{off-diagonal processor}\end{cases}$$
$$\Delta_{rot}=38\ cycles$$
$$T_{cyc}=269\ cycles$$
$$T_{ite}=\Delta_{ang}+(\frac{n}{2}-2)\delta_{pass}+T_{rot}=317+\frac{n}{2}\ cycles$$
$$T_{swe}=(n-2)T_{cyc}+T_{ite}=269.5n-221\ cycles$$
$$T_{svd}=(S(n-1)-1)T_{cyc}+T_{ite}$$
$$=269S(n-1)+\frac{n}{2}-48\ cycles$$

The utilization factor is

$$U_{C3}=\frac{\left(468\times\frac{n}{2}+112\times\left(\left(\frac{n}{2}\right)^2-\frac{n}{2}\right)\right)\times(n-1)\times S}{\left(4\times\frac{n}{2}+2\times\left(\left(\frac{n}{2}\right)^2-\frac{n}{2}\right)\right)\times T_{svd}}$$

Parameter	on-line (radix-2)	conventional		
		Case 1	Case 2	Case 3
T_{cyc}	1	2.45	2.45	3.28
T_{ite}	1	1.60	1.73	2.56
T_{swe}	1	2.27	2.30	3.15
T_{svd}	1	2.42	2.43	3.21
G_{diag}	1	0.84	0.84	0.15
G_{off}	1	1.38	0.77	0.19
Total cost	1	1.16	0.77	0.18

Table 6: Performance and cost ratio against on-line scheme for $n = 8$ and $S = 10$

Operation	On-line	Conventional
Add	1337	4773
Multiply	3941	12063
Divide	4261	4993
Square root	4261	4993

Table 7: Cost comparison of on-line and conventional arithmetic unit design

For $n = 8$ and $S = 10$ we have $U_{C3} = 0.3$.

Table 6 shows the performance and cost ratio for the conventional and on-line SVD with $n = 8$ and $S = 10$, assuming no cancellation of leading digits. The figures also reflect the ratio between the step-times of the schemes.

The comparison shows that for $n = 8$, the on-line scheme achieves a speedup of about 2.4 over Case 2 of the conventional implementation and about 3.2 over Case 3, and the gate count ranging from factor 1.3 to 5.5 of the conventional approach. For each processor, the on-line scheme requires 80 data connections, while the conventional scheme needs 512 data connections.

5 Concluding remarks

This paper reports a preliminary results on the implementation cost and performance of on-line arithmetic with respect to conventional arithmetic for a particular technology (gate arrays) and a particular application (SVD). The system itself has not yet been fully implemented. Our estimates are based on the gate-level designs of individual on-line units and provide a good estimate of the system gate-count. The results indicate that for double precision floating-point computations, the designs of the on-line arithmetic units have less complexity than the conventional designs. Table 7 compares the gate counts of the designs. For the SVD computation, the on-line scheme has a much larger number of arithmetic units and a greater gate count than the conventional scheme. We note that because of the overlapping of the on-line operations, the hardware utilization of the on-line scheme is much higher than the conventional designs. Moreover, on-line approach reduces for each processor the communication bandwidth from 512 to 80 data connections which is of significance in system packaging.

The results presented here are limited in several respects. We did not analyze performance in the case of cancellation of leading digits. This is a serious problem especially for on-line arithmetic because of performance degradation. Cancellation can increase the on-line delay up to the working precision, i.e., 56. The numerical behavior and its relation to cost and performance of an SVD on-line arithmetic system need to be addressed. In our study of the system cost we included the necessary variable delay buffers to synchronize operands which get out of phase due cancellation. To reduce the buffering cost, an alternative is to have control with blocking mechanisms. We note that when operands to a floating-point on-line adder are generated by an on-line multiplier, the least significant half of the product is available in parallel form and, at a cost of two shift registers, it can be used to extend the number of significant digits to the adder. Another issue is the interface with a host system and mechanisms for significance monitoring in on-line algorithms.

References

[1] R. P. Brent and F. T. Luk. The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM Journal on Scientific and Statistical Computing*, 6(1):69–84, January 1985.

[2] J. R. Cavallaro and F. T. Luk. CORDIC arithmetic for an SVD processor. *Proceedings of the 8th Symposium on Computer Arithmetic*, pages 113–120, 1987.

[3] M. D. Ercegovac. On-line arithmetic: an overview. In *Proceedings of the SPIE, Real Time Signal Processing VII*, volume 495, pages 86–93, San Diego, CA, August 1984.

[4] M. D. Ercegovac and T. Lang. On-line schemes for computing rotation angles for SVDs. In *Proceedings of the SPIE, Advanced Algorithms and Architectures for Signal Processing II*, volume 826, pages 160–169, August 1987.

[5] M. D. Ercegovac and T. Lang. On-line arithmetic: A design methodology and applications in digital signal processing. In R. W. Brodersen and H. Moscovitz, editors, *VLSI Signal Processing, III*, chapter 24, pages 252–263. IEEE Press, 1988.

[6] M. D. Ercegovac and T. Lang. Redundant and on-line CORDIC: Application to matrix triangularization and svd. *IEEE Transactions on Computers*, 39(6):725–740, June 1990.

[7] J.-A. Lee. *Redundant CORDIC: Theory and its application to matrix computations.* PhD thesis, University of California, Los Angeles, 1990.

[8] J.-A. Lee and T. Lang. On-line CORDIC for generalized singular value decomposition. *Proceedings of the SPIE, High Speed Computing II*, pages 235–247, 1989.

[9] LSI Logic Corporation, 1551 McCarthy Blvd, Milpitas CA 95035. *Databook and Design Manual for HCMOS Macrocells and Macrofunctions*, October 1986.

[10] F. T. Luk. Architectures for computing eigenvalues and SVDs. *Proceedings of the SPIE, Highly Parallel Signal Processing Architectures*, 614:24–33, 1986.

[11] H. Park. *On the equivalence and convergence of parallel Jacobi SVD algorithms*. PhD thesis, Cornell University, August 1987.

[12] F. T. L. Richard P. Brent and C. V. Loan. Computation of the singular value decomposition using mesh-connected processors. *Journal of VLSI and Computer Systems*, 1(3):242–270, 1985.

[13] J. M. Speiser and H. J. Whitehouse. A review of signal processing with systolic arrays. In *Proceedings of the SPIE, vol. 431, Real Time Signal Processing VI*, pages 2–6, August 1983.

[14] G. S. Taylor. Radix 16 SRT dividers with overlapped quotient selection stages. In *Proceedings of the 7th Symposium on Computer Arithmetic*, pages 64–71, 1985.

[15] P. K.-G. Tu. *On-line arithmetic algorithms for efficient implementations*. PhD thesis, University of California, Los Angeles, 1990.

[16] P. K.-G. Tu and M. D. Ercegovac. Design of on-line division unit. In *Proceedings of the 9th Symposium on Computer Arithmetic*, pages 42–49, 1989.

[17] P. K.-G. Tu and M. D. Ercegovac. Gate array implementation of on-line algorithms for floating-point operations. In *Proceedings of the Twenty-Fourth Asilomar Conference on Signals, Systems and Computers*, 1990.

[18] VIEWlogic Systems, Inc., 313 Boston Post Road West, Marlboro, Massachusetts 01752. *WORKVIEW Manual Set, Release 3.0*, June 1988.

Appendix: Floating-Point Algorithms

A Radix-2 on-line addition

Step 1. [Initialization and alignment]
$e_d \leftarrow e_x - e_y$
$e_z \leftarrow \max(e_x, e_y) + 1$
$A[-2] = 0, \quad z_0 = 0$
for $j = 1, \cdots, |e_d|$ **do**
  if $e_d \geq 0$ **then**
    $x'_j = x_j, \quad y'_j = 0$
  else
    $x'_j = 0, \quad y'_j = y_j$

for $j = |e_d| + 1, \cdots, m$ **do**
  if $e_d \geq 0$ **then**
    $x'_j = x_j, \quad y'_j = y_{j-|e_d|}$
  else
    $x'_j = x_{j-|e_d|}, \quad y'_j = y_j$

for $j = -1, \cdots, 0$ **do**
  $A[j] \leftarrow 2A[j-1] + (x'_{j+2} + y'_{j+2})2^{-2}$

Step 2. [result generation]
for $j = 1, \cdots, m+1$ **do**
  $A[j] \leftarrow 2(A[j-1] - z_{j-1})$
    $+(x'_{j+2} + y'_{j+2})2^{-2}$
  /* $A[j]$ has 3 bits of precision */
  $z_j \leftarrow \begin{cases} -1 & \textbf{if } A[j] < -2^{-1} \\ 0 & \textbf{if } -2^{-1} \leq A[j] < 2^{-1} \\ 1 & \textbf{if } A[j] \geq 2^{-1} \end{cases}$
  $z_{out} \leftarrow \text{postnorm}_{add}(z_j, z_{j+1}), z_{m+2} = 0$

end.

The post-normalization for addition works as follows:

1. Examine the first two digits (z_1, z_2) of the result.
2. If the values of the digits being examined, (z'_1, z'_2), are (1,0), (1,1), (-1,0), or (-1,-1), then the result is quasi-normalized. These digits and the rest of the result are output in sequence.
3. Otherwise, we combine the two digits as

$$z'_1 = 2 \cdot z'_1 + z'_2 \tag{4}$$

   the result exponent is decremented, and we get the next result digit $z'_2 \leftarrow z_{next}$.
4. Steps 2 and 3 are repeated until either m result digits have been examined without satisfying the condition of step 2, in which case the output is zero, or the condition in step 2 is satisfied, in which case z'_1, z'_2 and the remaining result digits are output.

In the optimal case where the condition of step 2 is satified for the first 2 digits of the result, the total on-line delay is $\delta = 5$, where 1 cycle is nedded to calculate the difference of the exponents, 3 cycles are needed to generate the first digit of the result, and 1 cycle is needed to get the second result digit.

In the worst case when all digits are exausted and the output is zero, the total on-line delay is the total number of cycles needed to obtain all digits of the result, which is $\delta_{cancel} \leq m + 3$, where m is the working precision.

B Radix-2 on-line multiplication

Step 1. [Initialization]
$e_z \leftarrow e_x + e_y$
$X[-3] = 0, Y[-3] = 0, A[-3] = 0, z_0 = 0$
for $j = -2, \cdots, 0$ **do**
  $X[j] \leftarrow X[j-1] + x_{j+3}2^{-j-3}$
  $A[j] \leftarrow 2A[j-1] + (x_{j+3}Y[j-1] + y_{j+3}X[j])2^{-3}$
  $Y[j] \leftarrow Y[j-1] + y_{j+3}2^{-j-3}$

Step 2. [product generation]
for $j = 1, \cdots, m+3$ **do**
  $X[j] \leftarrow X[j-1] + x_{j+3}2^{-j-3}$
  $A[j] \leftarrow 2(A[j-1] - z_{j-1})$
    $+(x_{j+3}Y[j-1] + y_{j+3}X[j])2^{-3}$

$Y[j] \leftarrow Y[j-1] + y_{j+3}2^{-j-3}$

/* $\widehat{A[j]}$ has 3 bits */

$$z_j \leftarrow \begin{cases} -1 & \text{if } \widehat{A[j]} < -2^{-1} \\ 0 & \text{if } -2^{-1} \le \widehat{A[j]} < 2^{-1} \\ 1 & \text{if } \widehat{A[j]} \ge 2^{-1} \end{cases}$$

$z_{out} \leftarrow \text{postnorm}_{mul}(z_j, z_{j+1}), z_{m+4} = 0$

end.

The post-normalization for multiplication works as the following.

1. Examine the first two digits (z_1, z_2) of the result.
2. If the values of the digits being examined, (z_1', z_2'), are (1,0), (1,1), (-1,0), or (-1,-1), then the result is quasi-normalized. The digits being examined and the rest of the result are output in sequence.
3. Otherwise, we combine the two digits as

$$z_1' = 2 \cdot z_1' + z_2' \tag{5}$$

the result exponent is decremented, and we get the next result digit $z_2' \leftarrow z_{next}$.

4. Steps 2 and 3 are repeated until either the condition in step 2 is satisfied, when z_1', z_2' and the remaining result digits are output, or z_4 has been examined, in which case the output digits are $z_1', z_5, \cdots$ and the rest of the result digits.

In the optimal case where the condition of step 2 is satisfied for the first 2 digits of the result, the total on-line delay is $\delta_{min} = 5$, where 4 cycles are needed to generate the first digit of the result, and 1 cycle is needed to get the second result digit.

In the worst case there can be at most 3 leading zeros in the result. The total online delay is the total number of cycles needed to obtain the first four digits of the result, which is $\delta_{max} = 7$.

C Radix-2 on-line division

Step 1. [Initialization and shifting]

$e_q \leftarrow e_n - e_d + 1$

$$A[0] \leftarrow \sum_{i=1}^{3} n_i 2^{-i-1}$$

$$D[0] \leftarrow \sum_{i=1}^{4} d_i 2^{-i}$$

if $D[0] < 2^{-1}$ **then**

$D[0] \leftarrow 2 \cdot D[0] + d_{next}2^{-4}$;

$e_q \leftarrow e_q + 1$

$Q[0] \leftarrow 0; \quad q_0 \leftarrow 0$

Step 2. [quotient generation]

for $j = 1, \cdots, m+2$ **do**

$D[j] \leftarrow D[j-1] + d_{next}2^{-j-4}$

$A[j] \leftarrow 2(A[j-1] - q_{j-1}D[j]) + n_{next}2^{-4} - d_{next}Q[j-2]2^{-4}$

/* $\widehat{A[j]}$ has 6 bits */

$$q_j = \begin{cases} -1 & \text{if } \widehat{A[j]} < -\frac{1}{4} \\ 0 & \text{if } -\frac{1}{4} \le \widehat{A[j]} < \frac{3}{16} \\ 1 & \text{if } \widehat{A[j]} \ge \frac{3}{16} \end{cases}$$

$Q[j] \leftarrow Q[j-1] + q_j 2^{-j}$

$q_{out} \leftarrow \text{postnorm}_{div}(q_j, q_{j+1}), q_{m+3} = 0$

end.

The post-normalization for division works as follows.

1. Examine the first two digits (q_1, q_2) of the result.
2. If the values of the digits being examined, (q_1', q_2'), are (1,0), (1,1), (-1,0), or (-1,-1), then the result is quasi-normalized. The digits being examined and the rest of the result are output in sequence.
3. Otherwise, we combine the two digits as

$$q_1' = 2 \cdot q_1' + q_2' \tag{6}$$

the result exponent is decremented, and we get the next result digit $q_2' \leftarrow q_{next}$.

4. Steps 2 and 3 are repeated until either the condition in step 2 is satisfied, when q_1', q_2' and the remaining result digits are output, or q_3 has been examined, in which case the output digits are $q_1', q_4, \cdots$ and the rest of the result digits.

In the optimal case where the condition of step 2 is satisfied for the first 2 digits of the result, the total on-line delay is 6, where 5 cycles are needed to generate the first digit of the result, and 1 cycle is needed to get the second result digit.

In the worst case there can be at most 2 leading zeros in the generated result. The total online delay is the total number of cycles needed to obtain the first three digits of the result, which is $\delta_{max} = 7$.

D Radix-2 on-line square root

Step 1. [Initialization and shifting]

if e_x *odd* **then**

$$A[0] \leftarrow \sum_{i=1}^{2} x_i 2^{-i-1}$$

else

$A[0] \leftarrow x_1 2^{-3}$

$e_y \leftarrow \lfloor e_x/2 \rfloor + 1$

Step 2. [output generation]

for $j = 1, \cdots, m$ **do**

$A[j] \leftarrow 2 \cdot (A[j-1] - y_{j-1}(Y[j-2] + Y[j-1])) + x_{next}2^{-4}$

/* $\widehat{A[j]}$ has 7 bits */

/* $\mathcal{S}_{sqrt}$ is the selection function defined in [15]. */

$y_j \leftarrow \mathcal{S}_{sqrt}(\widehat{A[j]})$

$Y[j] \leftarrow Y[j-1] + y_j 2^{-j}$

end.