Proceedings

1989 IEEE INTERNATIONAL CONFERENCE ON Computer Design: VLSI in Computers & Processors

Sponsored by IEEE Computer Society and IEEE Circuits and Systems Society
In cooperation with IEEE Electron Devices Society

ICCD '89
Hyatt Regency Cambridge, Cambridge, Massachusetts
October 2-4, 1989

IEEE

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE
COMPUTER
SOCIETY
PRESS

Fault-Tolerant VLSI Processor Array for the SVD

Joseph R. Cavallaro

Dept. of Electrical & Computer Engineering
Rice University
Houston, TX 77251

Christopher D. Near

M. Ümit Uyar

AT&T Bell Laboratories
Holmdel, NJ 07733

Abstract

Dynamic reconfiguration techniques are presented for a two-dimensional systolic array for the SVD of a matrix. Extra computation time is not required, since idle time inherent in the array is exploited. This scheme does not require additional spare processors and is easily implemented in VLSI. Only minor hardware and communication time increases within each processing element are required.

1. Introduction

The Singular Value Decomposition (SVD) of a matrix is an important numerical algorithm used extensively in real-time signal processing and image processing applications. Special purpose parallel processor arrays for computing the SVD can significantly improve the performance of this computationally intensive algorithm. The systolic array architecture of Brent, Luk, and Van Loan [3] uses an expandable square array of processors to compute the SVD of a large matrix. Recently, the high-speed CORDIC arithmetic algorithms have been applied to the SVD [5] and a VLSI implementation of a processor array element is in progress at Rice University.

A major application of a CORDIC SVD array is space and underwater systems where the processor cannot be physically repaired. In this paper, a fault tolerant CORDIC SVD array which uses dynamic reconfigurability is presented. Faults are considered permanent rendering a processor and its associated communication links unusable.

The techniques reported in the literature for fault reconfiguration require additional processors to replace the faulty ones and can only handle a limited number of faults before the array becomes unusable [2, 10]. In the fault reconfiguration technique presented here, the computation load of the faulty processors is handled by its neighbors. This scheme requires no additional processors and handles $O(p)$ faulty processors in a $\lceil p/2 \rceil \times \lceil p/2 \rceil$ array operating on a $p \times p$ matrix. Utilizing idle time inherent in the SVD algorithm, the fault reconfiguration scheme requires no additional computation time. However, the extra communication time of an array with faulty processors leads to 5% more total time than the fault-free case. Although the processor hardware is increased by 5%, no additional communication links are needed. Dynamic reconfiguration is a

difficult problem and the subject of extensive research. This work is an attempt to provide an algorithm for the SVD array.

2. The Singular Value Decomposition Array

The singular value decomposition of an $p \times p$ matrix M is

$$M = U \Sigma V^T,$$

where U and V are orthogonal matrices and Σ is a diagonal matrix of singular values. In the Brent, Luk, Van Loan array, the matrix is divided into 2×2 submatrices. Each processor element contains a 2×2 submatrix. The array architecture is scalable. As an example, Figure 1 shows a small array to solve a 6×6 problem. There are

Figure 1. Data and Angle Flow of the Brent-Luk-Van Loan SVD method.

In the CORDIC SVD array, the processor is active once in every three major cycles. The idle second and third major cycles are necessary for the systolic propagation of the angle data and the associated rotations. Within each major cycle, there are three CORDIC minor cycles. At an even lower level, there are n "micro" cycles which correspond to the shifts and additions that compose a CORDIC iteration. Data transmission between processors will contain n -bit values. The assumption is made that one "micro" cycle, (the time for an n -bit addition), will be more than sufficient to transmit data to an immediate neighbor.

In our fault model, only processor and/or link faults which render a processor and its associated links unusable are considered. Various fault detection schemes may be employed to locate and isolate the faults, including self-implicating structures [7], voting schemes based on redundancy [9], algorithmic checksum techniques [1], and special SVD fault detection schemes [6]. Detection of the faults is not addressed in this paper.

Once a fault has been detected, the faulty processor(s) and/or links are isolated and the array is dynamically reconfigured. A traditional approach to dynamic reconfiguration has been to view the mesh as a graph and then perform a cut to remove the entire row, column, or diagonal with the faulty processor. The remaining processors are rejoined and processing resumes after the algorithm has been modified to deal with the new dimensions of the mesh. This scheme has been used in the Massively Parallel Processor (MPP) [2] and has been discussed by Kung and Lam [10]. These redundancy-based fault tolerance techniques are also reviewed by Negrini, Sami, and Stefanelli [11]. However, only a few errors can be tolerated before the mesh is reduced to an unusable size.

In this paper, a more versatile fault reconfiguration technique which removes just the faulty processor and reassigns the faulty processor's data to the fault-free neighbors by a uniform data redistribution [13] is introduced. Reconfiguration through uniform data redistribution has been developed for a variety of algorithms and interconnection networks [8, 12]. In this scheme, the faulty processor's computations are performed by neighboring processors and interprocessor communication is then routed around the location of the faulty processor. For example, if a diagonal CORDIC SVD processor is faulty, its data will be assigned to its neighbor for computation.

Each processor in the array can operate in either the fault-free, (FF), or fault-tolerant, (FT), mode. Each proces-

If it is a near-neighbor to a faulty processor, then it modifies its communication pattern to talk to different neighbors around the faulty processor. In addition, if it is the neighbor which is to perform the additional computation, then it uses one of its idle cycles to operate on the faulty processor's data.

In the event of a fault, a pre-determined neighbor is assigned to process the 2×2 matrix of data points of the faulty processor. The neighbor is chosen in order to minimize the communication time, since it is close to the source of the angles passed between the processors. An alternative path is established around the faulty processor to preserve the systolic transmission of the rotation angles and the interchange of matrix data elements, since the communication links to the faulty processor are also assumed to be faulty.

177

Figure 2b. Matrix Data Routing Around a Super-Diagonal Fault.

The actual sequence of events for super-diagonal processors in the CORDIC SVD array is illustrated in Table 1. Timing for sub-diagonal processors is similar; however, the direction of angle and data flow is reversed. Communication and computation events are shown for each processor. Each column of the table refers to a diagonal band of processors as shown in Figures 2a and 2b. There are two critical timing events: the rotation angle transfer and the matrix data transfer. The rotation angle transfer will control the synchronization of the systolic array since computation in any processor element occurs directly after the receipt of the rotation angles. The rotation angles are generated in the diagonal processors of the array. These angles propagate horizontally and vertically and require a transmission, T_{Te} , and reception, T_{Re} , time.

The second critical timing event in the systolic array is the transmission, T_{TD} , and reception, T_{RD} , of matrix data elements. This data is exchanged diagonally between processors. For example, a processor in Diagonal 3 requires four data elements. The last data element to arrive is from the Northeast neighbor on Diagonal 5. The processor on Diagonal 3 then applies the rotation angles and computes new matrix data elements. The results are passed to the diagonal neighbors and in particular to the Southwest neighbor on Diagonal 1.

In developing a fault reconfiguration strategy, it is important that the backup processor is near the source of angles and data, and that it is idle during the cycle. The processor which is shown as faulty in Figures 2a and 2b is on Diagonal 3. In Table 1, the west processor in Diagonal 2 will perform the computation during the idle period marked *FT*. Two extra data transfer steps (not shown in Table 1) will also be needed to complete the data communication.

Table 1
Timing Events in a CORDIC SVD Processor Array

D1	D2	D3	D4	D5
T_{RD} T_{Re} T_e T_D T_{TD} T_{RD} T_{Te}	T_{RD} T_{Re} T_e T_D T_{TD} T_{RD} T_{Te} *FT* T_{RD} T_{Re}	T_{RD} T_{Re} T_{RD} T_{Re} T_e T_D T_{TD} T_{RD} T_{Te}	T_{RD} T_{Re} T_e T_D T_{TD} T_{RD} T_{Te}	T_{RD} T_{Re} T_e T_D T_{TD} T_{RD} T_{Te}
T_{RD} T_{Re} T_e T_D T_{TD} T_{RD} T_{Te}	T_{RD} T_{Re}	T_{RD} T_{Re} T_e T_D T_{TD} T_{RD} T_{Te}	T_{RD} T_{Re} T_e T_D T_{TD} T_{RD} T_{Te}	T_{RD} T_{Re}
T_{Re} Receive rotation angles T_{Te} Transmit rotation angles T_{RD} Receive matrix data T_{TD} Transmit matrix data T_e Compute rotation angles (CORDIC cycle) T_D Compute new matrix data (2 CORDIC cycles) *FT* Processor acts as Backup Processor for D3				

The CORDIC SVD processor array is toroidally connected in order to allow greater reconfiguration flexibility. Since the array contains $\lceil p/2 \rceil \times \lceil p/2 \rceil$ processors, the following arithmetic is modulo $\lceil p/2 \rceil$. The following scheme is used to determine the backup processor in the event of a single fault:

Diagonal Processors:

For all P_{ii} : Backup is West neighbor, $P_{i,i-1}$.

Super-Diagonal Processors:

For P_{ij} where $j > i$: Backup is West neighbor, $P_{i,j-1}$.

Sub-Diagonal Processors:

For P_{ij} where $i > j$: Backup is North neighbor, $P_{i-1,j}$.

The case of more than one disjoint fault (ie., faulty processors that are not neighbors) is also handled in this manner. This scheme can handle many disjoint faults within the array, since hardware is not consumed in the reconfiguration. No conflicts will occur near the main diagonal of the array, since faults are assumed to be disjoint. However, adjacent faulty processors may require a more difficult fault reconfiguration scheme.

5. Fault Tolerant Architectural and Processor Requirements

In order to bypass the faulty processors, the 8-connected mesh architecture required by the Brent-Luk-Van Loan SVD algorithm will be exploited. Therefore, there are no additional communication links required for the dynamic fault reconfiguration scheme.

5.1. Communication Time Costs

The fault free CORDIC SVD processor time, T_{FF_CSVD} , contains two major components, communication, T_{COMM} , and computation, T_{COMP} , time. Analysis of the computation time [5] has shown that $T_{COMP} = 3.25T_C$, where $T_C = n(T_{ADD})$ for an n -bit implementation and n -bit addition. Since idle time within the array is exploited in the fault reconfiguration scheme, the computation time does not increase. However, extra communication steps are needed because of the rerouting of data on the critical timing path. The extra communication time will include two extra data transfers, T_{DT} . Each of the data transfer steps shown in Table 1, T_{R0} , T_{T0} , T_{RD} , and T_{TD} are assumed to be equal to T_{DT} . In the fault free case, $T_{COMM} = 5T_{DT}$. Based upon the performance of 2μ CMOS I/O pads, $T_{DT} < T_{ADD}$, and therefore $T_{COMM} < T_{COMP}$. The total fault free time is:

$$T_{FF_CSVD} = T_{COMP} + T_{COMM} = 3.25T_C + 5T_{DT}.$$

The total time for a 2×2 fault-tolerant CORDIC SVD, T_{FT_CSVD} , will be:

$$T_{FT_CSVD} = T_{FF_CSVD} + 2T_{DT}.$$

For a typical value of n , $n = 32$, the increase in time is less than 5%.

5.2. Hardware Costs

In order for the processors to handle the extra data of a faulty near neighbor, additional hardware is required. Each processor will require 6 extra registers, 4 for data

and 2 for angles. Also, a multiplexor will be used to handle the two different data sets. The fault free area, A_{FF_CSVD} , is:

$$A_{FF_CSVD} = 2A_C + A_{CONT},$$

where A_{CONT} is the area of the SVD control programmable logic array (PLA), and the area of a CORDIC processor is $A_C = A_{CPLA} + A_{ROM} + 2A_{ASH} + 3A_{ADD} + 3A_{REG}$. The various areas correspond to the internal CORDIC PLA, ROM, barrel shifter, adder, and registers, respectively.

The extra hardware units, shown in Figure 3, will include the 6 registers in the CORDIC processors, labelled FT , plus a bus communication controller, A_{FT_BC} , for the fault-tolerant communication protocol. This controller will be implemented as a PLA. The area of a fault tolerant CORDIC SVD processor, A_{FT_CSVD} is:

$$A_{FT_CSVD} = A_{FF_CSVD} + 6A_{FT_REG} + A_{FT_BC}.$$

Considering the complexity of the adders and barrel shifters required by a non-reconfigurable SVD processor [4], the extra hardware for fault reconfiguration is less than 5%.

6. Summary

An efficient scheme for dynamic fault reconfiguration for the CORDIC SVD processor array is presented. This scheme does not require either additional processors or communication links. The procedure utilizes the idle time

Figure 3. Block Diagram of Fault Tolerant CORDIC SVD processor array element.

inherent in the CORDIC SVD algorithm to achieve fault tolerance with only 5% additional time. Each processor will have an increase in hardware complexity of 5%. The design of this fault reconfiguration method preserves the array regularity necessary for VLSI implementation. A VLSI prototype implementation of this scheme is in progress at Rice University.

References

1. Abraham, J. A., Banerjee, P., Chen, C. -Y., Fuchs, W. K., Kuo, S. -Y., and Reddy, A. L. N., "Fault Tolerance Techniques for Systolic Arrays," *IEEE Computer*, 20,(7), pp. 65-75 (July 1987).
2. Batcher, K. E., "Design of a Massively Parallel Processor," *IEEE Transactions on Computers*, C-29,(9), pp. 836-840 (September 1980).
3. Brent, R. P., Luk, F. T., and Van Loan, C. F., "Computation of the Singular Value Decomposition Using Mesh-Connected Processors," *Journal of VLSI and Computer Systems*, 1,(3), pp. 242-270 (1985).
4. Cavallaro, J. R., "VLSI CORDIC Processor Architectures for the Singular Value Decomposition," Ph. D. Dissertation, School of Electrical Engineering, Cornell Univ., Ithaca, NY (August 1988).
5. Cavallaro, J. R. and Luk, F. T., "CORDIC Arithmetic for an SVD Processor," *Journal of Parallel and Distributed Computing*, pp. 271-290 (June 1988).
6. Chen, C. Y. and Abraham, J. A., "Fault-Tolerant Systems for the Computation of Eigenvalues and Singular Values," *Proc. SPIE Architectures and Algorithms for Signal Processing*, 696,(1), pp. 228-237, San Diego, CA (August 1986).
7. Dahbura, A. T., Masson, G. M., and Yang, C.-L., "Self-Implicating Structures for Diagnosable Systems," *IEEE Transactions on Computers*, C-34,(8), pp. 718-723 (August 1985).
8. Elster, A. C., Uyar, M. Ü., and Reeves, A. P., "Fault-Tolerant Matrix Operations on Hypercube Multiprocessors," *IEEE Int'l Conference on Parallel Processing*, in press, St. Charles, IL (August 1989).
9. Emmerson, R. and McGowan, M. J., "Fault Tolerance Achieved in VLSI," *IEEE Micro*, pp. 34-43 (December 1984).
10. Kung, H. T. and Lam, M. S., "Fault-Tolerant VLSI Systolic Arrays and Two-Level Pipelining," *Journal of Parallel and Distributed Computing*, 1,(1), pp. 32-63 (1984).
11. Negrini, R., Sami, M. G., and Stefanelli, R., *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays*, MIT Press, Cambridge, MA (1989).
12. Uyar, M. Ü. and Reeves, A. P., "Fault-Tolerant Bitonic Sorting in Hypercubic MIMD Systems," Technical Report, School of Electrical Engineering, Cornell Univ., Ithaca, NY (February 1986).
13. Uyar, M. Ü. and Reeves, A. P., "Dynamic Fault Reconfiguration in a Mesh-Connected MIMD Environment," *IEEE Trans. on Computers*, C-37,(10), pp. 1191-1205 (October 1988).