AD-A191 770

REPORT DOCUMENTATION PAGE

UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	Approved for public release; distribution is unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Naval Ocean Systems Center	6b. OFFICE SYMBOL (if applicable)
6c. ADDRESS (City, State and ZIP Code): San Diego, CA 92152-5000	7a. NAME OF MONITORING ORGANIZATION
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Office of Naval Research	8b. OFFICE SYMBOL (if applicable): ONR
8c. ADDRESS (City, State and ZIP Code): 800 North Quincy Street, Arlington, VA 22217	7b. ADDRESS (City, State and ZIP Code)
	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO	TASK NO.	AGENCY ACCESSION NO.
61153N	EE55	RR01509	DN088 669

11. TITLE (Include Security Classification): Architecture of the Systolic Linear Algebra Parallel Processor (SLAPP)

12. PERSONAL AUTHOR(S): J.J. Symanski

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Journal Article	FROM _____ TO _____	August 1986	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	real-time signal processing integer or floating point numbers

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper will present preliminary concepts for the design of a systolic array of processors specifically aimed at efficient implementation of a core set of matrix operations consisting of matrix multiplication, ZRD, SVD and generalized SVD. The algorithms to be implemented will be discussed briefly. Concepts for efficient implementation of the algorithms will be presented along with future plans.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
J.J. Symanski	619-225-2200	Code 741

DD FORM 1473, 84 JAN — 83 APR EDITION MAY BE USED UNTIL EXHAUSTED ALL OTHER EDITIONS ARE OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

To appear in the Proceedings of the SPIE International Technical Symposium, Vol. 698-34, Real Time Signal Processing, San Diego, CA 17-22 August 1986.

Architecture of the Systolic Linear Algebra Parallel Processor (SLAPP)

J. J. Symanski

Naval Ocean Systems Center
San Diego, CA 92152

Abstract

This paper will present preliminary concepts for the design of a systolic array of processors specifically aimed at efficient implementation of a core set of matrix operations consisting of matrix multiplication, QRD, SVD and generalized SVD. The algorithms to be implemented will be discussed briefly. Concepts for efficient implementation of the algorithms will be presented along with future plans.

Introduction

The importance of the QRD, SVD and GSVD to real-time signal processing has been discussed by Speiser[1]. These algorithms are far more complex than the FFT and similar algorithms which require basically only multiplications and additions of integer or floating point numbers. H. T. Kung's[2] elegant and efficient concept of data rhythmically flowing through linear or two-dimensional arrays of processors, becomes more complicated with these advanced algorithms. This added complexity is partly due to the algorithms and partly to the state-of-the-art of integrated circuits available to construct these processors. Divisions and square roots, are more difficult and time consuming to achieve in hardware, since they require iterative approaches or additional complex circuits.

Other factors which lead to implementation difficulties are data movement and the programming of long pipelines of data. For applications where the data arrays have dimensions on the order of 100 to 1000 and the dynamic range of floating point computations is required, it is currently infeasible to build arrays with one processor for each data element. This means that techniques must be found to map large data arrays onto smaller physical arrays and maintain a high level of efficiency for the algorithms.

This paper will present a new architecture, implementable with current state-of-the-art integrated circuits which attempts to efficiently implement the core operations of matrix multiplication, the QR, SVD and GSVD as described by Luk[3,4], as well as containing a high level of flexibility for the implementation of other algorithms and application dependent functions. The architecture attempts to deal with the problems of latency in the primitive arithmetic operations, imbalances in data transfer and computation rates, processing of arrays of data larger than the physical array and some software issues.

Algorithms

The algorithms of primary interest are the QRD, SVD and GSVD of Luk[3,4]. The GSVD is similar to the SVD but more complex in that it will be implemented on two triangular arrays which will make its data movements more complex. The GSVD is currently being analyzed with respect to its mapping onto the array. Matrix multiplication is relatively easy to implement on the array and will be discussed only briefly.

The QRD algorithm of Luk[2,3] is similar to that of Gentlemen and Kung[6] in that it is based on a triangular array of processors. However, it is organized to permit a smoother data flow when used for both a QRD and SVD or GSVD. The general appearance of Luk's computational network is shown in Figure 1, where the data elements are shown as squares and the processors are shown as circles. This is a conceptual representation only.

Luk's array consists of $(n \times n)/4 + 0(n)$ processors and a triangular array of storage cells located around the processors. (In our implementation these storage cells are brought into the processor, since they are easily incorporated into RAM memory in the processor.) The algorithm basically operates on 2 x 2 matrices along the diagonal to obtain rotation sines and cosines which are then applied along the rows to annihilate the leading element of each row as it comes into the array from above. This process continues until the m elements of the input data matrix have been processed and we are left with a n x n triangular matrix. (This is a common situation in signal processing, where we have m time samples from n sensors producing an array of m x n data elements.)

The SVD uses a similar triangular array of $(n \times n)/4 + 0(n)$ processors, as shown in figure 2. The input matrix must be in the upper triangular form which can be obtained by using the QRD. The approach is to use the diagonal processors of the array to annihilate

the off diagonal elements via Jacobi rotations. We compute block 2 x 2 SVDs along the main diagonal and pass rotation sines and cosines to the right and up to eliminate internal elements. For a given 2 x 2 block this is a two step process in which the 2 x 2 matrix is first symmetrized and then diagonalized. Then, an odd-even ordering scheme is applied to the 2 x 2 blocks. First, the odd index blocks are reduced to diagonal form, data passed, and then the even numbered blocks are similarly reduced. This process proceeds until $n(n-1)/2$ transformations have been completed. This is one sweep of the algorithm. Luk[2] proposes that iterations be stopped after about ten sweeps, which is usually sufficient to achieve convergence.

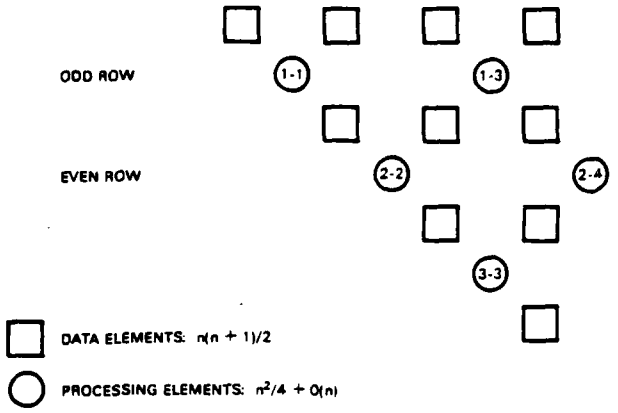

Figure 1. The Luk QRD Computational Network

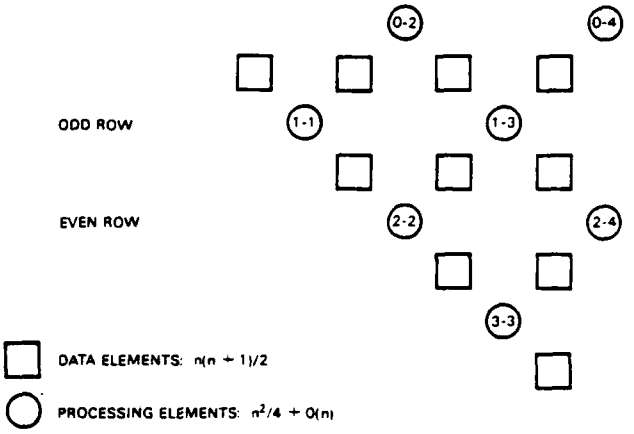

Figure 2. The Luk SVD Computational Network

The GSVD will utilize this same triangular array but will require two arrays with a data path between corresponding elements of the two triangular arrays. The details of this mapping will be reported in a later paper.

We propose to implement the algorithms on a triangular array of processors as shown in Figure 3. The processors on the diagonal, shown as circles, are called boundary processors and are designed to perform the generation of sines and cosines required for the Jacobi rotations, quickly with low latency. Note that trigonometric functions are not explicitly computed. Generation of the sines and cosines requires only square roots, divisions, multiplies and additions. The off diagonal or interior processors, shown as squares, are similar but do not (necessarily) have the capability to generate sines and cosines quickly. All processors have the capability to perform multiplications and additions with 32 bit floating point data.

Matrix multiplication is a very regular and simple algorithm to implement in a square array. Since two triangular arrays will be required for the GSVD the two triangular arrays will be connected in such a way as to form a square array with two main diagonals, or an (n+1) x n array, as shown in Figure 4. Matrix multiplication will be accomplished in a manner similar to that described by Symanski[7].

Architecture

The architecture proposed can be thought of simply as two triangular arrays similar to that shown in Figure 3, placed parallel to each other and connected with data paths between corresponding elements in each array. This results in a three-dimensional array of two triangular planes. Two triangular arrays are required for the solution of the GSVD and also to gain the factor of n when performing matrix multiplications. Some applications require as much computation in matrix multiplications as in the more complex QR and SVD. The availability of n squared processors speeds the throughput of matrix multiplies by a factor of n which becomes significant as n increases.

The architecture of the processing element is shown in Figure 5. It consists of an Input/Output Processor (IOP) connected via a dual-port RAM to a Linear Algebra Processor (LAP). There is an auxiliary RAM module for temporary data storage and also interprocessor communications circuitry to enable the IOP to queue tasks for the LAP.

The key concept here is that the IO is independent of the computation in any processing element. As long as the IO overhead is low, we can make a gain in overall throughput by sharing computational tasks among processors. This makes programming of an algorithm or a set of algorithms for specific application much easier since the programmer does not have

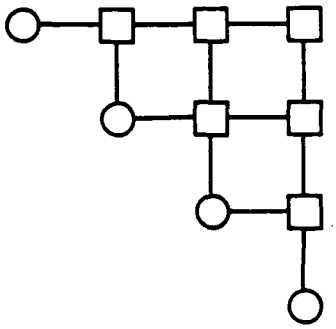

Figure 3. The Triangular Systolic Array

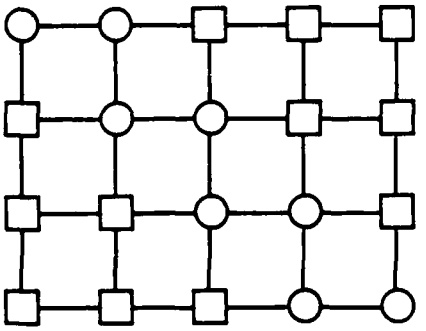

Figure 4. Dual Triangular Arrays Configured for Matrix Multiplication

to worry about interruptions in the flow of the algorithm. This is especially advantageous with the pipelined floating point units currently available for implementing these processors.

Furthermore, this could be extended to any nearest (or even not so nearest neighbors) as long as there are 'available cycles' in a processing element and the IO doesn't wipe out the time gained by using other processors. The whole game is to keep as many of the processors as busy as possible and obtain as close to a linear gain in processing for all of the array, as possible.

The LAP block diagram is shown in Figure 6. The floating point multiplier/accumulator is a device such as the Weitek 3332. This device performs multiplies, adds, data conversions and also contains a 32 word register file. The Unary Function Module, which is currently under development, performs arithmetic functions of one variable, such as inverse, square root, etc.

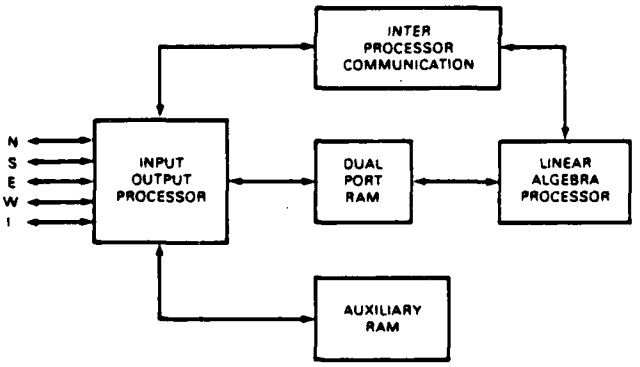

Figure 5. The SLAPP Processing Element

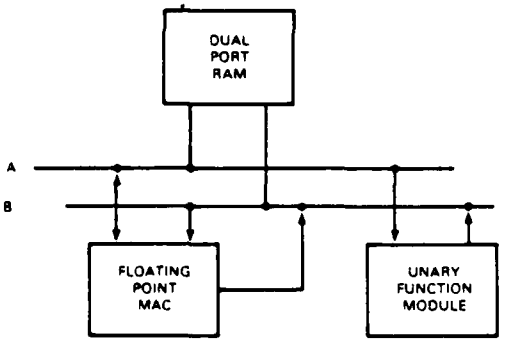

Figure 6. The Linear Algebra Processor

Unary Function Module

There are many instances in the computations of the algorithms of interest where we require a unary function, i.e., a function of a single variable, for instance the inverse of x. The use of a high speed unary function module can significantly speed up the computation of the rotation sines and cosines, thus cutting down the latency of the boundary processors.

Previous work by Nowatzyk[8] at Carnegie-Mellon University indicated that the basic functions of the inverse of x, the square root of x and the inverse of the square root of x could be obtained using about thirty TTL integrated circuits. The results could be obtained in about 150 nanoseconds. During our analysis of the arithmetic functions needed for the QRD and the SVD, other functions were found which would also be useful. Table 1 shows the list to date. Some of these functions are difficult to obtain, requiring larger tables and some additional logic. Others are trivial, but may be useful if the availability will reduce programming complexity and computation latency.

The block diagram for the unary function circuit is shown in Figure 7. This module is in the early stages of design so only a simplified discussion of its operation can be given here. Input to the module will be a 32-bit IEEE format floating point number and function codes to specify the desired output. The 32-bit IEEE format result will have latency of two or three 100 nanosecond clock cycles.

Table 1. Unary Functions Useful in Linear Algebra

Inverse x
Square root x
Inverse square root x
Square root $(1 + x^2)$
Inverse square root $(1 + x^2)$
Sign x and -Sign x
2x and -2x
Abs x and Abs 2x
+Sine θ and -Sine θ [sign $x/\sqrt{1 + x^2}$]

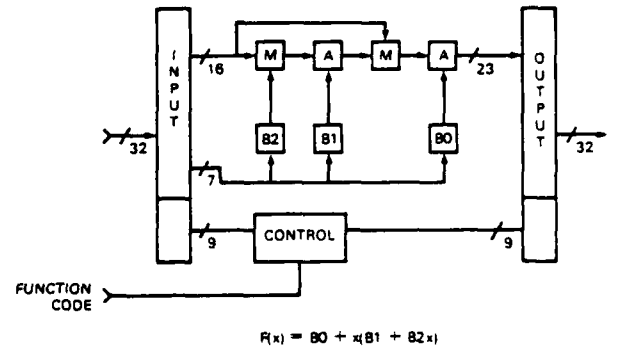

Figure 7. Unary Function Block Diagram

The exponent and control codes are input to the control circuit which decodes the function code and controls the rest of the module circuitry and generates the correct exponent. The mantissa is separated into two parts. The most significant seven bits plus some control lines go to the ROMS B0, B1 and B2 which produce seed values for the computation. The least significant sixteen bits are feed to two 16 x 16 multipliers. Two 28-bit ALUs sum these products to produce the mantissa of the result.

Extensive analysis and simulation indicates that the least significant bit of the result will have an error rate of less than 6 percent. Details of this work will be presented in a later report.

Future Plans

There are many details to be studied and verified. This architecture promises a new level of parallelism but at a cost of complexity of the computations and data movement. The GSVD has to be analyzed as to its operation and mapping onto the array. The prime objective is to fully utilize the available bandwidth of the processors. The approach of separating the IO from the computation has advantages in programming of the array and allowing different processors to perform similar or very different processes simultaneously, similar to a MIMD array. This also has implications for fault tolerance through reconfiguration and redistribution of the processing load.

The next step is to verify the efficient operation of the algorithms through detailed analysis and simulation of the operation of the array and individual processing elements. Throughput for typical problems will be determined. High level modeling will suffice to accomplish this. Design can then proceed to detailed logic definition and simulation, which will be done on an engineering workstation.

Acknowledgments

This work is funded by Dr. Richard L. Lau of the Office of Naval Research, Code 1111. Program management was ably performed by Dr. Keith Bromley.

The concepts presented are the result of discussions with many people. Harper Whitehouse first recognized the signal processing implications of Luk's QRD and SVD techniques and proposed the dual triangular architecture for the SLAPP. Co-workers Barry Drake and Jeffrey Speiser at NOSC were of great help in understanding the algorithms. John Celto and Tom Hendersen of NOSC have done considerable work on the unary function module. Special thanks go to Frank Luk and researchers at Cornell for helpful information on the algorithms. The author also gratefully acknowledges many stimulating discussions with Professor H. T. Kung and members of the WARP project at Carnegie-Mellon University.

References

1. Jeffrey Speiser, "Linear Algebra Algorithms for Matrix-Based Signal Processing," Highly Parallel Signal Processing Architectures, SPIE Critical Review of Technology, Los Angeles, CA, Jan 1986, SPIE Volume 614, paper 614-01.

2. H. T. Kung, "Why Systolic Architectures?," Computer, IEEE Society, Volume 15, Number 1, January 1982.

3. Franklin T. Luk, "A Triangular Processor Array for Computing the Singular Value Decomposition," Cornell University Department of Computer Science Technical Report TR 84-625, July 1984.

4. Franklin T. Luk, "Architectures for Computing Eigenvalues and SVDs," Technical Report EE-CEG-86-1, February 1986, Cornell University, Ithaca, New York, 14853.

5. Franklin T. Luk, "A Parallel Method for Computing the Generalized Singular Value Decomposition," Technical Report EE-CEG-85-1, January 1985, Cornell University, Ithaca, New York, 14853.

6. W. M. Gentlemen, and H. T. Kung, "Matrix Triangularization by Systolic Arrays," Proc. SPIE, Vol. 298, Real-time Signal Processing IV, Tien F. Tao, Editor, SPIE, 1981.

7. J. J. Symanski, "Implementation of Matrix Operations on the Two-dimensional Systolic Array Testbed", Proceedings of the SPIE International Technical Symposium, San Diego, CA, 21-26 August 1983.

8. Andreas Nowatzyk, "Fast Evaluation of Arithmetic Functions," Carnegie-Mellon University, Computer Science Department, Technical Report CMU-CS-85-169, September 1985.