Parallel Algorithms for Digital Signal Processing

Richard P. Brent
Computer Sciences Laboratory
Australian National University
Canberra, Australia

1. Introduction

This paper provides an introduction to some parallel algorithms relevant to digital signal processing. First we introduce some basic concepts such as *speedup* and *efficiency* of parallel algorithms. We also outline some practical parallel computer architectures – pipelined, SIMD and MIMD machines, hypercubes and systolic arrays.

To illustrate the basic concepts and key issues, we consider the problem of parallel solution of a nonsingular linear system. Gaussian elimination with pivoting is difficult to implement on many parallel architectures, but omission of pivoting leads to numerical instability. A solution is to implement a parallel version of the orthogonal (QR) decomposition instead of the triangular (LU) decomposition obtained by Gaussian elimination. We consider as an application the solution of linear least squares problems.

Many problems in digital signal processing are easy to solve if we can find the singular value decomposition (SVD) of a rectangular matrix, or the eigenvalues and eigenvectors of a symmetric (or Hermitian) matrix. We describe some good parallel algorithms for these problems. Often the parallel algorithms are based on old ideas (such as Jacobi's method for finding the eigenvalues of a symmetric matrix) rather than on a straightforward adaption of the best serial algorithms. The parallel algorithms can be implemented efficiently on systolic arrays, but we also mention how they might be implemented on other parallel architectures.

Toeplitz systems often arise in digital signal processing. However, we do not discuss their solution here as they are considered in the companion paper [6].

2. Basic concepts

We assume that a parallel machine with P processors is available. Thus P measures the degree of parallelism; $P = 1$ is just the familiar serial case. When considering the solution of a particular problem, we let T_P denote the time required to solve the problem using (at most) P processors. The *speedup* S_P is defined by

$$S_P = T_1/T_P,$$

and the *efficiency* $E_P = S_P/P$.

Appeared in *Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms* (edited by G. H. Golub and P. Van Dooren), Springer-Verlag, 1991, 93–110.
 rpb110 typeset using TEX

When converting a serial algorithm into a parallel algorithm, our aim is usually to attain constant efficiency, i.e.

$$E_P \geq c$$

for some positive constant c independent of P. This may be written as

$$E_P = \Omega(1).$$

Equivalently, we want to attain linear speedup, i.e.

$$S_P \geq cP,$$

which may be written as

$$S_P = \Omega(P).$$

2.1 Amdahl's Law

Suppose a positive fraction f of a computation is “essentially serial”, i.e. not amenable to any speedup on a parallel machine. Then we would expect

$$T_P = fT_1 + (1-f)T_1/P$$

so the overall speedup

$$S_P = \frac{1}{f + (1-f)/P} \leq \frac{1}{f}, \tag{2.1}$$

i.e. the speedup is *bounded*, not linear. The inequality (2.1) is called *Amdahl's Law* and is often used as an argument against parallel computation. However, what it shows is that the speedup is bounded as we increase the number of processors for a *fixed* problem. In practice, it is more likely that we shall want to solve larger problems as the number of processors increases.

Let N be a measure of the problem size. For many problems it is reasonable to assume that

$$f \leq K/N \tag{2.2}$$

for some constant K. For example, in problems involving N by N matrices, we may have $\Omega(N^3)$ arithmetic operations and $O(N^2)$ serial input/output.

Suppose also that N increases at least linearly with P, i.e.

$$N \geq KP. \tag{2.3}$$

(2.2) and (2.3) imply that $fP \leq 1$, so from (2.1) we have

$$S_P = \frac{P}{fP + (1-f)} \geq \frac{P}{2-f} \geq \frac{P}{2}.$$

Thus we get linear speedup, with efficiency $E_P \geq 1/2$.

2.2 Virtual processors

In practice any parallel machine has a fixed maximum number (say P) of processors imposed by hardware constraints. When analysing parallel algorithms it is often convenient to ignore this fact and assume that we have as many (say p) processors as desired. We may think of these p processors as *virtual processors* or *processes.* Each *real processor* can simulate $\lceil p/P \rceil$ virtual processors (provided each real processor has enough memory). Thus, ignoring overheads associated with the simulation, we have

$$S_P \geq S_p / \lceil p/P \rceil. \tag{2.4}$$

If our analysis for p processors gives us a lower bound on S_p, then (2.4) can be used to obtain a lower bound on S_P.

2.3 Parallel architectures

Many varieties of parallel computer architecture have been proposed in recent years. They include –

- Pipelined vector processors such as the Cray 1 or Cyber 205, in which there is a single instruction stream and the parallelism is more or less hidden from the programmer.
- Single-instruction multiple-data (SIMD [16]) machines such as the Illiac IV or ICL DAP, in which a number of simple processing elements (PEs or cells) execute the same instruction on local data and communicate with their nearest neighbours on a square grid or torus. There is usually a general-purpose controller which can broadcast instructions and data to the cells.
- Multiple-instruction multiple-data (MIMD) machines such as transputer systems, C.mmp, CM*, and most hypercube machines.
- Hypercube machines, in which 2^k processors are connected like the vertices of a k-dimensional cube (i.e. if the processors are identified by k-bit binary numbers, they are connected to the processors whose numbers differ by exactly one bit from their own). These include both SIMD machines (e.g. the Connection Machine) and MIMD machines (e.g. the Caltech "Cosmic Cube", the Intel iPSC, and the NCUBE).
- Shared-memory multiprocessors such as the Alliant FX/8, the Encore Multimax, and the Sequent Symmetry.
- Systolic arrays [30], which are 1 or 2-dimensional arrays of simple processors (cells) connected to their nearest neighbours. The cells on the edge of the array are usually connected to a general-purpose machine which acts as a controller. Examples are the Warp and iWarp machines [1, 5] and several machines described in [43]. Variations on the idea of systolic arrays are wavefront arrays [37] and instruction systolic arrays [45].

In view of the diversity of parallel computer architectures, it is difficult to describe practical parallel algorithms in a machine-independent manner. In some cases an algorithm intended for one class of parallel machine can easily be converted for another (more general) class. For example, an algorithm designed for a systolic array can be mapped onto a hypercube without much loss of efficiency, but not conversely (in general). An algorithm designed for a local memory machine can easily be implemented on a shared memory machine, but often the converse is not true.

Since systolic arrays are very restrictive, it is usually possible to map systolic array algorithms onto other parallel machines without a great loss in efficiency. On the other hand, systolic arrays are sufficient and cost-effective for problems arising in digital signal processing [30, 36, 37, 43]. Thus, we shall generally describe parallel algorithms as though they are to be implemented on systolic arrays – the reader should be able to translate to other parallel computer architectures.

3. Systolic algorithms for signal processing

Many compute-bound computations with applications in signal processing have good parallel algorithms which can be implemented on systolic arrays. For example, we mention:

- 1-D convolution, FIR and IIR filtering [11, 28, 29, 30]
- 2-D convolution and correlation [2, 32, 33, 34, 50]
- discrete Fourier transform [28, 29]
- interpolation [32]
- 1-D and 2-D median filtering [15]
- matrix-vector and matrix-matrix multiplication [31, 48]
- solution of triangular linear systems [31]
- LU and QR factorization of square matrices, and solution of full-rank linear systems [4, 19, 31]
- QR factorization of rectangular matrices, and solution of linear least squares problems [19, 38]
- solution of the symmetric and Hermitian eigenvalue problems [3, 7, 8]
- computation of the singular value decomposition (SVD) [9, 10, 39, 40, 44]
- solution of Toeplitz linear systems and least squares problems [6].

In Section 4 we consider a "simple" example – the solution of square, full-rank linear systems. Then, in Section 5, we consider the computation of the SVD and the solution of the symmetric eigenvalue problem.

4. The solution of linear systems

Suppose we want to solve a nonsingular n by n linear system

$$Ax = b$$

on a systolic array or other parallel machine for which a 2-dimensional mesh is a natural interconnection pattern. It is easy to implement Gaussian elimination *without* pivoting, because multipliers can be propagated along rows of the augmented matrix $[A|b]$, and it is not necessary for one row operation to be completed before the next row operation starts. Unfortunately, as is well-known [23, 47, 49], Gaussian elimination without pivoting is numerically unstable unless A has some special property such as diagonal dominance or positive definiteness.

On a serial machine the problem of numerical instability could easily be overcome by using partial (or complete) pivoting. However, pivoting is difficult to implement efficiently on a parallel machine because it destroys the regular data flow associated with Gaussian elimination without pivoting [31]. Just to find the first pivot (i.e. $\max_{1\le i\le n}|a_{i,1}|$) takes time $\Omega(n)$ on a systolic array ($\Omega(\log n)$ on a hypercube), so the complete elimination takes time $\Omega(n^2)$ on a systolic array ($\Omega(n\log n)$ on a hypercube).

If $T_P = \Omega(n^2)$ we are not justified in using more than $P = O(n)$ processors (else $T_P P \gg n^3$ and the efficiency $E_P \ll 1$).

On a linear systolic array of $P = n + 1$ processors, we store one column of the augmented matrix $[A|b]$ in each processor, and we can implement partial pivoting efficiently. If $P \gg n$ it is not so easy to implement partial pivoting efficiently. One solution is to replace the elementary transformations

$$\begin{pmatrix} 1 & 0 \\ -m & 1 \end{pmatrix}$$

of Gaussian elimination by plane rotations (Givens transformations)

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix},$$

where $c^2 + s^2 = 1$. Thus, we construct an *orthogonal* matrix Q such that $QA = R$ is upper triangular and the linear system $Ax = b$ is reduced to the upper triangular system $Rx = \bar{b}$ (where $\bar{b} = Qb$) in a *numerically stable* manner. The operation count is roughly four times that for Gaussian elimination, but this is an acceptable price to pay for numerical stability on a parallel machine. The factor four could be reduced by the use of "fast" Givens transformations [18]. A closely related alternative, with similar data flow requirements but inferior numerical properties, is to use "pairwise pivoting", i.e. pivoting between adjacent rows.

4.1 Implementation of parallel Givens – the BBK *algorithm*

To simplify communication on a systolic array or mesh-connected parallel machine it is desirable to restrict attention to plane rotations which modify *adjacent* rows. Since each rotation modifies only two rows, it is possible to do up to $\lfloor n/2\rfloor$ rotations in parallel. One of many ways to order the rotations necessary to reduce A to upper triangular form was suggested by Bojanczyk, Brent and Kung (BBK) [4] and is illustrated below (where $n = 8$ and the numbers in the illustration indicate the time step at which the corresponding matrix element is zeroed):

$$\begin{pmatrix}
\cdot & & & & & & & \\
7 & \cdot & & & & & & \\
6 & 9 & \cdot & & & & & \\
5 & 8 & 11 & \cdot & & & & \\
4 & 7 & 10 & 13 & \cdot & & & \\
3 & 6 & 9 & 12 & 15 & \cdot & & \\
2 & 5 & 8 & 11 & 14 & 17 & \cdot & \\
1 & 4 & 7 & 10 & 13 & 16 & 19 & \cdot
\end{pmatrix}$$

The rotation parameters (c, s) propagate right along rows and it is not necessary to wait for one row transformation to be completed before others commence. Details of the implementation on a triangular systolic array may be found in Bojanczyk, Brent and Kung [4].

4.2 Solution of least squares problems – the Gentleman and Kung algorithm

The BBK algorithm was intended for the solution of the n by n linear system $Ax = b$. When applied to the m by n linear least squares problem

$$min\|Ax - b\|_2$$

it is inefficient because it uses a triangular systolic array of $m^2/2 + O(m)$ processors, and usually in such problems $m \gg n$. Gentleman and Kung [19] (and later Luk [38]) transposed the data flow, so the skewed matrix A is fed into the systolic array by columns rather than by rows. This is advantageous because a triangular systolic array of only $n^2/2 + O(n)$ processors is required. In fact, it is not even necessary to know m in advance when using the Gentleman-Kung algorithm.

A good survey of systolic array algorithms for computing the QR decomposition of a (square or) rectangular matrix may be found in Luk [38].

5. The SVD and symmetric eigenvalue problems

A singular value decomposition (SVD) of a real m by n matrix A is its factorization into the product of three matrices:

$$A = U\Sigma V^T, \tag{5.1}$$

where U is an m by n matrix with orthonormal columns, Σ is an n by n nonnegative diagonal matrix, and V is an n by n orthogonal matrix (we assume here that $m \geq n$). The diagonal elements σ_i of Σ are the *singular values* of A. The singular value decomposition is extremely useful in digital signal processing [21, 36].

The SVD is usually computed by a two-sided orthogonalization process, e.g. by two-sided reduction to bidiagonal form (possibly preceded by a one-sided QR reduction [12]) followed by the QR algorithm [20, 22, 49]. On a systolic array it is simpler to avoid bidiagonalization and to use the two-sided orthogonalization method of Kogbetliantz et al [9, 10, 17, 26, 27] rather than the standard Golub-Kahan-Reinsch algorithm [20, 22]. However, it is even simpler to use a one-sided orthogonalization method due to Hestenes [25]. The idea of Hestenes is to iteratively generate an orthogonal matrix V such that AV has orthogonal columns. Normalizing the Euclidean length of each nonnull column to unity, we get

$$AV = \tilde{U}\Sigma \tag{5.2}$$

As a null column of $\tilde{U}$ is always associated with a zero diagonal element of Σ, there is no essential difference between (5.1) and (5.2).

There is clearly a close connection between the Hestenes method for finding the SVD of A and the classical Jacobi method [49] for finding the eigenvalues and eigenvectors of A^TA. This is discussed in Section 5.4.

5.1 Implementation of the Hestenes method

Let $A_1 = A$ and $V_1 = I$. The Hestenes method uses a sequence of plane rotations Q_k chosen to orthogonalize two columns in $A_{k+1} = A_k Q_k$. If the matrix V is required, the plane rotations are accumulated using $V_{k+1} = V_k Q_k$. Under certain conditions (discussed below) $\lim Q_k = I$, $\lim V_k = V$ and $\lim A_k = AV$. The matrix A_{k+1} differs from A_k only in two columns, say columns i and j. In fact

$$\left(a_i^{(k+1)}, a_j^{(k+1)}\right) = \left(a_i^k, a_j^k\right) \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

where the rotation angle θ is chosen so that the two new columns $a_i^{(k+1)}$ and $a_j^{(k+1)}$ are orthogonal. This can always be done with an angle θ satisfying

$$|\theta| \le \pi/4, \tag{5.3}$$

see for example [8].

It is desirable for a "sweep" of $n(n-1)/2$ rotations to include all pairs (i, j) with $i < j$. On a serial machine a simple strategy is to choose the "cyclic by rows" ordering

$$(1, 2), (1, 3), \cdots, (1, n), (2, 3), \cdots, (n-1, n).$$

Forsythe and Henrici [17] have shown that the cyclic by rows ordering and condition (5.3) ensure convergence of the Jacobi method applied to $A^T A$, and convergence of the cyclic by rows Hestenes method follows.

5.2 The chess tournament analogy

On a parallel machine we would like to orthogonalize several pairs of columns simultaneously. This should be possible so long as no column occurs in more than one pair. The problem is similar to that of organizing a round-robin tournament between n players. A game between players i and j corresponds to orthogonalizing columns i and j, a round of several games played at the same time corresponds to orthogonalizing several pairs of (disjoint) columns, and a tournament where each player plays each other player once corresponds to a sweep in which each pair of columns is orthogonalized. Thus, schemes which are well-known to chess (or other two-person game) players can be used to give orderings amenable to parallel computation. It is usually desirable to minimize the number of parallel steps in a sweep, which corresponds to the number of rounds in the tournament.

On a parallel machine with restricted communication paths, such as a systolic array, there are constraints on the orderings which we can implement efficiently. A useful analogy is a tournament of lazy chess players. After each round the players want to walk only a short distance to the board where they are to play the next round.

Using this analogy, suppose that each chess board corresponds to a systolic processor and each player corresponds to a column of the matrix (initially A but modified as the computation proceeds). A game between two players corresponds to orthogonalization of the corresponding columns. Thus we suppose that each systolic processor has sufficient memory to store and update two columns of the matrix. If the chess boards (processors) are arranged in a linear array with nearest-neighbour communication paths, then the players should have to walk (at most) to an adjacent board between the end of one round and the beginning of the next round, i.e. columns of the matrix should have to be exchanged only between adjacent processors. Several orderings satisfying these conditions have been proposed [7, 8, 41, 44].

Since A has n columns and at most $\lfloor n/2 \rfloor$ pairs can be orthogonalized in parallel, a sweep requires as least $n-1$ parallel steps (n even) or n parallel steps (n odd). The "Brent-Luk" ordering [8] attains this minimum, and convergence can be guaranteed if n is odd [41, 46]. It is an open question whether convergence can be guaranteed for the Brent-Luk ordering (or any other ordering which requires only the minimum number of parallel steps) when n is even – the problem in proving convergence is illustrated in [24]. However, in practice lack of convergence is not a problem, and it is easy to ensure convergence by the use of a "threshhold" strategy [49], or by taking one additional parallel step per sweep when n is even [42].

5.3 A heuristic argument regarding the number of sweeps

In practice we observe linear convergence until the columns of the (updated) matrix A become close to orthogonal, i.e. until the off-diagonal elements of A^TA become small. Subsequently convergence is at least quadratic. It is difficult to quantify what "small" means here unless the eigenvalues of A^TA are distinct. If we assume that "small" means $O(1/n^c)$ for some positive constant c, and that the rate of linear convergence is independent of n, then it should take $O(\log n)$ sweeps before quadratic convergence sets in. This heuristic argument is supported by empirical evidence [8] – certainly the average number of sweeps required to give convergence to a fixed tolerance for random matrices A appears to be $O(\log n)$.

5.4 The symmetric eigenvalue problem

As noted above, there is a close connection between the Hestenes method for finding the SVD of a matrix A and the Jacobi method for finding the eigenvalues of a symmetric matrix $B = A^TA$. An important difference is that the formulas defining the rotation angle θ involve elements $b_{i,j}$ of B rather than inner products of columns of A, and transformations must be performed on the left and right instead of just on the right (since $(AV)^T(AV) = V^TBV$). Instead of permuting columns of A as described in Section 5.2, we have to apply the same permutation to both rows and columns of B. This is easy if we use a square systolic array of $n/2$ by $n/2$ processors with nearest-neighbour connections (assuming, for simplicity, that n is even). Details are given in [8].

If less than $n^2/4$ processors are available, we can use the virtual processor concept described in Section 2.2. For example, on a linear systolic array with $P \le n/2$ processors, each processor can simulate $\sim n/(2P)$ columns of $n/2$ virtual processors. Similarly, on a square array of $P \le n^2/4$ processors, each processor can simulate a block of $\sim n^2/(4P)$ virtual processors. In both cases, communication paths between virtual processors map onto paths between real processors or communication internal to a real processor.

5.5 Other SVD and eigenvalue algorithms

In Section 5.2 we showed how the Hestenes method could be used to compute the SVD of an m by n matrix in time $O(mn^2S/P)$ using $P = O(n)$ processors in parallel. Here S is the number of sweeps required (conjectured to be $O(\log n)$). In Section 5.4 we sketched how Jacobi's method could be used to compute the eigen-decomposition of a symmetric n by n matrix in time $O(n^3S/P)$ using $P = O(n^2)$ processors. It is natural to ask if we can use more than $\Omega(n)$ processors efficiently when computing the SVD. The answer is yes – Kogbetliantz [26, 27] and Forsythe & Henrici [17] suggested an analogue of Jacobi's method, and this can be used to compute the SVD of a *square* matrix using a parallel algorithm very similar to the parallel implementation of Jacobi's method. The result is an algorithm which requires time $O(n^3S/P)$ using $P = O(n^2)$ processors. Details and a discussion of several variations on this theme may be found in [9].

In order to find the SVD of a rectangular m by n matrix A using $O(n^2)$ processors, we first compute the QR factorization $QA = R$ (see Section 4), and then compute the SVD of the principal n by n submatrix of R (i.e. discard the $m - n$ zero rows of R). It is possible to gain a factor of two in efficiency by preserving the upper triangular structure of R [39].

The Hestenes/Jacobi/Kogbetliantz methods are not often used on a serial computer, because they are slower than methods based on reduction to bidiagonal or tridiagonal form followed by the QR algorithm [49]. Whether the fast serial algorithms can be implemented efficiently on a parallel machine depends to some extent on the parallel architecture. For example, on a square array of n by n processors it is possible to reduce a symmetric n by n matrix to tridiagonal form in time $O(n \log n)$ [3]. On a serial machine this reduction takes time $O(n^3)$. Thus, a factor $O(\log n)$ is lost in efficiency, which roughly equates to the factor $O(S)$ by which Jacobi's method is slower than the QR algorithm on a serial machine. It is an open question whether the loss in efficiency by a factor $O(\log n)$ can be avoided on a parallel machine with $P = \Omega(n^2)$ processors. When $P = O(n)$, "block" versions of the usual serial algorithms are attractive on certain architectures [13], and may be combined with the "divide and conquer" strategy [14]. Generally, these more complex algorithms are attractive on shared memory MIMD machines with a small number of processors, while the simpler algorithms described in Sections 5.1 to 5.4 are attractive on systolic arrays and SIMD machines.

References

1. M. Annaratone, E. Arnould, T. Gross, H. T. Kung, M. Lam, O. Menzilcioglu and J. A. Webb, "The Warp computer: architecture, implementation and performance", *IEEE Transactions on Computers*, C-36 (1987), 1523-1538.
2. J. Blackmer, P. Kuekes and G. Frank, "A 200 MOPS systolic processor", Proc. SPIE, Vol. 298, *Real-Time Signal Processing IV*, Society of Photo-Optical Instrumentation Engineers, 1981.
3. A. W. Bojanczyk and R. P. Brent, "Tridiagonalization of a symmetric matrix on a square array of mesh-connected processors", *J. Parallel and Distributed Computing* 2 (1985), 261-276.

4. A. W. Bojanczyk, R. P. Brent and H. T. Kung, *Numerically stable solution of dense systems of linear equations using mesh-connected processors*, Tech. Report TR-CS-81-119, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., May 1981. Also appeared in *SIAM J. Scientific and Statistical Computing* 5 (1984), 95-104.
5. S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski and J. Webb, "iWarp: An integrated solution to high-speed parallel computing", *Proc. Supercomputing 1988 Conference*, Orlando, Florida, Nov. 1988, 330-339.
6. R. P. Brent, "Parallel algorithms for Toeplitz systems", these proceedings.
7. R. P. Brent, H. T. Kung and F. T. Luk, "Some linear-time algorithms for systolic arrays", in *Information Processing 83* (edited by R.E.A. Mason), North-Holland, Amsterdam, 1983, 865-876.
8. R. P. Brent and F. T. Luk, "The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays", *SIAM J. Scientific and Statistical Computing* 6 (1985), 69-84.
9. R. P. Brent, F. T. Luk and C. F. Van Loan, "Computation of the singular value decomposition using mesh-connected processors", *J. of VLSI and Computer Systems* 1, 3 (1983-1985), 242-270.
10. R. P. Brent, F. T. Luk and C. F. Van Loan, "Computation of the generalized singular value decomposition using mesh-connected processors", *Proc. SPIE, Volume 431, Real Time Signal Processing VI*, Society of Photo-Optical Instrumentation Engineers, Bellingham, Washington, 1983, 66-71.
11. P. R. Cappello and K. Steiglitz, "Digital signal processing applications of systolic algorithms", in [35], 245-254.
12. T. F. Chan, "An improved algorithm for computing the singular value decomposition", *ACM Trans. Math. Software* 8 (1982), 72-83.
13. J. J. Dongarra, S. J. Hammarling and D. C. Sorensen, "Block reduction of matrices to condensed forms for eigenvalue computations", *LAPACK Working Note #2*, MCS Division, Argonne National Labs., Argonne, Illinois, Sept. 1987.
14. J. Dongarra and D. Sorensen, "A fully parallel algorithm for the symmetric eigenproblem", *SIAM J. Scientific and Statistical Computing* 8 (1987), 139-154.
15. A. Fisher, "Systolic algorithms for running order statistics in signal and image processing", in [35], 265-272.
16. M. J. Flynn, "Some computer organizations and their effectiveness", *IEEE Transactions on Computers*, C-21 (1972), 702-706.
17. G. E. Forsythe and P. Henrici, "The cyclic Jacobi method for computing the principal values of a complex matrix", *Trans. Amer. Math. Soc.* 94 (1960), 1-23.
18. W. M. Gentleman, "Least squares computations by Givens transformations without square roots", *J. Inst. Math. Appl.* 12 (1973), 329-336.
19. W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic arrays", *Proc. SPIE, Volume 298, Real-Time Signal Processing IV*, Society of Photo-Optical Instrumentation Engineers, 1981.
20. G. H. Golub and W. Kahan, "Calculating the singular values and pseudo-inverse of a matrix", *J. SIAM Ser. B: Numer. Anal.* 2 (1965), 205-224.
21. G. H. Golub and F. T. Luk, "Singular value decompostion: applications and computations", ARO Report 77-1, *Trans. 22nd Conference of Army Mathematicians*,

1977, 577-605.

22. G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions", *Numer. Math.* 14 (1970), 403-420.
23. G. H. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins Press, Baltimore, Maryland, 1983.
24. E. R. Hansen, "On cyclic Jacobi methods", *J. SIAM* 11 (1963), 448-459.
25. M. R. Hestenes, "Inversion of matrices by biorthogonalization and related results", *J. SIAM* 6 (1958), 51-90.
26. E. Kogbetliantz, "Diagonalization of general complex matrices as a new method for solution of linear equations", *Proc. Internat. Congress of Mathematicians*, Amsterdam, Vol. 2, 1954, 356-357.
27. E. Kogbetliantz, "Solution of linear equations by diagonalization of coefficient matrices", *Quart. Appl. Math.* 13 (1955), 123-132.
28. H. T. Kung, "Let's design algorithms for VLSI systems", *Proc. Conf. on Very Large Scale Integration: Architecture, Design, Fabrication*, California Institute of Technology, Pasadena, Jan. 1979, 65-90.
29. H. T. Kung, "Special-purpose devices for signal and image processing: an opportunity in VLSI", *Proc. SPIE, Volume 241, Real-Time Signal Processing III*, Society of Photo-Optical Instrumentation Engineers, July 1980, 76-84.
30. H. T. Kung, "Why systolic architectures ?", *IEEE Computer* 15, 1 (1982), 37-46.
31. H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)", *Sparse Matrix Proc. 1978*, SIAM, Philadelphia, 1979, 256-282.
32. H. T. Kung and R. L. Picard, "Hardware pipelines for multi-dimensional convolution and resampling", *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, IEEE, Nov. 1981, 273-278.
33. H. T. Kung, L. M. Ruane and D. W. L. Yen, "A two-level pipelined systolic array for convolutions", in [35], 255-264.
34. H. T. Kung and S. W. Song, *A systolic 2-D convolution chip*, Technical Report CMU-CS-81-110, Carnegie-Mellon University, Pittsburgh, March 1981.
35. H. T. Kung, R. F. Sproull and G. L. Steele, Jr. (eds.), *VLSI Systems and Computations*, Computer Science Press, 1981.
36. S. Y. Kung (editor), *VLSI and Modern Signal Processing*, Proceedings of a Workshop held at Univ. of Southern California, Nov. 1982.
37. S. Y. Kung, *VLSI Array Processors*, Prentice-Hall International, 1988.
38. F. T. Luk, "A rotation method for computing the QR-decomposition", *SIAM J. Scientific and Statistical Computing* 7 (1986), 452-459.
39. F. T. Luk, "A triangular processor array for computing singular values", *J. of Linear Algebra and its Applications* 77 (1986), 259-273.
40. F. T. Luk, "Architectures for computing eigenvalues and SVDs", *Proc. SPIE, Vol. 614, Highly Parallel Signal Processing Architectures*, 1986, 24-33.
41. F. T. Luk and H. Park, "On parallel Jacobi orderings", *SIAM J. Scientific and Statistical Computing* 10 (1989), 18-26.
42. F. T. Luk and H. Park, "A proof of convergence for two parallel Jacobi SVD algorithms", *IEEE Transactions on Computers* 30 (1989), 806-811.
43. W. Moore, A. McCabe and R. Urquhart (editors), *Systolic Arrays*, Adam Hilger, Bristol, 1987.

44. D. E. Schimmel and F. T. Luk, "A new systolic array for the singular value decomposition", *Proc. Fourth MIT Conference on Advanced Research in VLSI*, 1986, 205-217.
45. H. Schröder, "The instruction systolic array – A tradeoff between flexibility and speed", *Computer Systems Science and Engineering* 3 (1988), 83-90.
46. G. Shroff and R. Schreiber, *On the convergence of the cyclic Jacobi method for parallel block orderings*, Report 88-11, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York, May 1988.
47. G. W. Stewart, *Introduction to Matrix Computations*, Ac. Press, New York, 1973.
48. U. Weiser and A. Davis, "A wavefront notation tool for VLSI array design", in [35], 226-234.
49. J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon, Oxford, 1965.
50. D. W. L. Yen and A. V. Kulkarni, "The ESL systolic processor for signal and image processing", *Proc. 1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Nov. 1981, 265-272.