Abstract.

The most significant impact on research in Scientific Computation, and Numerical Linear Algebra in particular, seems to have been brought about by the advent of vector and parallel computation. This paper presents a short survey of recent work on parallel implementations of Numerical Linear Algebra algorithms with emphasis on those relating to the solution of the symmetric eigenvalue problem on loosely coupled multiprocessor architectures.

A simple model will be given to analyse the complexity of parallel algorithms on several representative multiprocessor systems : a linear processor array (or ring), a two-dimensional processor grid and the hypercube. The vital operations in the formulation of most eigenvalue algorithms are matrix vector multiplication, matrix transposition, and linear system solution. Their implementations on the above architectures will be described, as well as parallel implementations of the following classes of eigenvalue methods : QR, bisection, divide-and-conquer, and Lanczos algorithm.

The Impact of Parallel Architectures on the Solution of Eigenvalue Problems

Ilse C.F. Ipsen, Youcef Saad

Research Report YALEU/DCS/RR-444
December 1985

The work presented in this paper was supported by the Office of Naval Research under contracts N00014-82-K-0184 and N00014-85-K-0461.

1. Introduction

Undoubtedly the most significant impact on research in Scientific Computation, and Numerical Linear Algebra in particular, has come about with the advent of vector and parallel computation. This paper presents a short survey of recent work on parallel implementations of Numerical Linear Algebra algorithms with emphasis on those relating to the solution of the symmetric eigenvalue problem on loosely coupled multiprocessor architectures.

Although the concept of parallel computation was well understood in the early years of electronic computing (already Babbage recognised it as a powerful means for speeding up the multiplication of two numbers [4]), it intermittently had to give way to the largely sequential von Neuman Computer. The recent turn to parallel computer architectures is motivated by the serious limitations inherent in the von Neuman model, the most important one being its limits to miniaturisation, imposed by physical constraints, which put a bound on the maximum speed of a logical circuit. Consequently, the only means for increasing computing speed by *orders* of magnitude is brought about by the use of parallel machines.

In a parallel computing device (also called 'multiprocessor') different processors share the computations involved in the solution of a problem. To this end, the computation must be decomposed and broken up into tasks, which can be performed simultaneously by the different processors; and organised co-operation among the processors must be established by means of synchronisation and data exchange. The selection of an algorithm, its decomposition into separate computational tasks and their subsequent assignment to particular processors, as well as the physical channels and protocols by means of which the processors communicate are among the many factors leading to a multitude of parallel implementations for any one problem.

The above decisions are aggravated by the need (or perhaps absence) of adequate performance measures. Even if a certain multiprocessor machine is already specified, one still faces the problem of having to decompose a particular algorithm into tasks with the objective of gaining maximal speed-up and a balanced work-load for all processors. Before that, however, reliable criteria for evaluating and comparing the performance of different implementations of an algorithm on that machine are indispensable. It is also necessary, of course, to be able to compare implementations of different algorithms on *one* machine, as well as implementations of different algorithms on *different* machines. These issues are far from being resolved. One of the reasons is that a fair assessment of two architectures must be based on the availability of adequate hardware as well as software. Yet, due to the absence of systematic design techniques the development of software is and will undoubtedly continue to lag behind hardware development.

Section 2 presents a brief characterisation of popular parallel architectures and justifies our preference for a loosely coupled multiprocessor architecture. The choice of machine model in turn influences the choice of a parallel algorithm for solving a particular problem. Extensive surveys of parallel algorithms can be found in the articles by Heller [35], Sameh [69, 70, 71], and Ortega and Voigt [60]. Unlike in the single processor case the performance of a parallel algorithm is not only judged by its arithmetic speed but, equally, by the time required to exchange data and coordinate co-operation among processors; understanding of this aspect has only started [29, 30, 61]. Accordingly, we present a collection of parallel algorithms for basic Linear Algebra tasks in Section 3 and their application to the parallel solution of eigenvalue problems in Section 4. To conclude, the last section casts a glance at some novel techniques which promise to alleviate the complex problem of parallel algorithm development.

2. Architectures

There exist quite a few classifications of multiprocessor architectures, and we will employ two of them. The first one distinguishes architectures by the way processors relate their instructions to the data while the second one groups machines according to the structure of their communication environment.

The two most important categories of parallel architectures are *Single Instruction Stream Multiple Data Stream* (SIMD) and *Multiple Instruction Stream Multiple Data Stream* (MIMD) machines [23]. SIMD machines initiate a single stream of vector instructions, which may be realised by pipelining in one processor or operating arrays of processors [38]. Examples include the CRAY-1, the ICL-DAP [22] and the ILLIAC IV [80]. MIMD machines simultaneously initiate different instructions on (necessarily) different data streams, essentially all multiprocessor configurations are included in this class [38].

Among the MIMD machines one can in turn differentiate between two types :

- *Shared memory models* : processors have very little local or 'private' memory; they exchange data and co-operate by accessing a global shared memory.
- *Distributed memory models* : there is no global memory, but processors possess a significant amount of local memory (with no access to other processor's local memory); there are physical interconnections between certain pairs of processors, and data and control information is transferred from one processor to another along a path of these interconnections.

2.1. The Shared Memory Model

The shared memory model is frequently implemented by connecting k processors to k memories via a large switching network, see Figure 1 (this switching network may be replaced by a global bus when the number of processors k is small). Thus the memory can be viewed as split into k 'banks', and shared among the k processors. Variations on this scheme are numerous, but the essential features here are the switching network and the shared memory; examples include the Ultracomputer developed at NYU [32] which uses an Omega network. Programming is greatly facilitated due to transparent data access (from the user's point of view data are stored in *one* large memory readily accessible to any processor) and the ability of the switching network to simulate any interconnection topology. However, memory conflicts can lead to degraded performance and the shared memory models cannot easily take advantage of proximity of data in problems with local (data) dependences; these questions are addressed in [25, 79]. Furthermore, the switching network becomes exceedingly complex as the number of processors and memories increases : the connection of N processors to N memories in general requires a total of $O(N \log_2 N)$ identical 2×2 switches.

2.2. The Distributed Memory Model

In the distributed memory model, the processors are identical and the processor interconnections form a regular topology; examples are depicted in Figures 2, 3 and 4. There is no tight global synchronisation, and the computations are data driven (ie, a computation in a particular processor is performed only when the needed operands become available). Examples include the finite element machine [47], tree machines [12], the cosmic cube [78] and systolic arrays [51].

Clearly, one of the most important advantages of the second class of architectures is its ability to exploit locality of data dependences in order to keep communication costs to a minimum. Thus, a two-dimensional processor grid as in Figure 3 is perfectly suitable for solving discretised elliptic partial differential equations (eg, by assigning each grid point to a corresponding processor) because iterative methods for solving the resulting linear systems require only interaction between adjacent grid points. Hence, an efficient general purpose multiprocessor must have powerful *mapping capabilities*, ie, it must be able to easily emulate many common topologies such as grids or linear arrays.

2.3. Hypercube-based Architectures

The 'hypercube' (boolean cube, n-cube), a distributed memory machine, constitutes an excellent compromise between a linear array and a completely interconnected network of processors. It

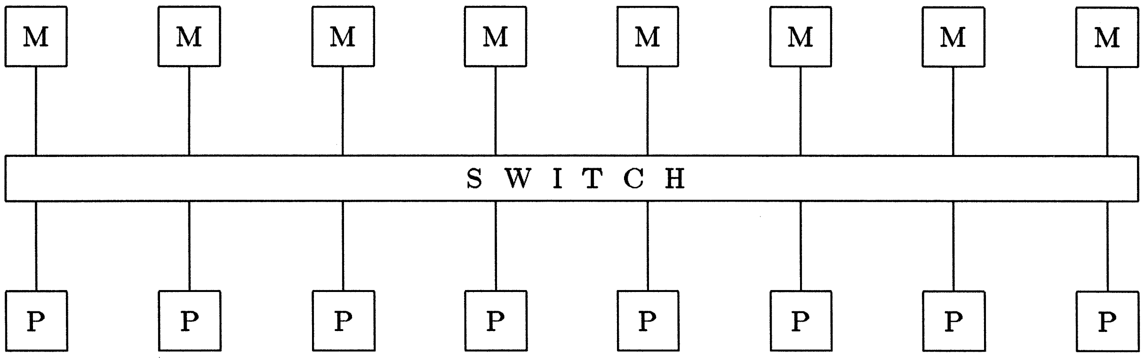

Figure 1: A Tightly Coupled Shared Memory Machine.

offers a rich interconnection structure with large bandwidth, logarithmic diameter, and the ability to simulate every realistic architecture with small overhead. This explains the growing interest in hypercube-based parallel machines; commercially available machines at this point (last quarter of 1985) are the 128-processor INTEL iPSC/d7, the 1024-processor NCUBE/Ten, the 64000-processor (bit-sliced) Connection Machine from Thinking Machines and the soon-to-be-available 256-processor Ametek/System 14 [19].

The topology of a hypercube is best described by a simple recursion : a hypercube of dimension 1 consists of two connected processors and a hypercube of dimension $n + 1$ is made up of two identical subcubes of dimension n by connecting processors in corresponding positions; an illustration is given in Figure 4 which shows a four-dimensional cube constructed from two three-dimensional cubes. Topological characterisations of the hypercube, in particular with respect to embeddings of different graphs in the hypercube are investigated in [6, 67, 65]

3. Parallel Algorithms for Basic Linear Algebra Computations

When analysing the complexity of parallel methods in numerical linear algebra, one must bear in mind that the total time required to run an algorithm on a multiprocessor system does not only depend on pure arithmetic time but also on the time needed for exchanging data among processors. This implies a great richness in the class of algorithms, in terms of the assignments of tasks to processors and the assumed topology of the processor communication network : for a particular task it is now important

- *when* its input data become available as results of preceding calculations,
- in *which* processor they are located, and
- how *long* it will take to move them to the requesting processor.

In many practical applications the number of processors k will usually be much smaller than the 'problem size' N (eg, the order of the matrix), and a large variety of algorithms can be found by choosing different ways of assigning matrix elements to the processors.

We must take into account that times for data transfer are *not* negligible and may, in fact, dominate the times for actual arithmetic. A fairly general and yet simple communication model is proposed in [43], and the algorithms are characterised and compared with respect to their requirements for arithmetic as well as communication.

It is assumed, that any processor is capable of writing to one of its directly connected neighbours while reading from the other. For purposes of estimating the computation time, processors

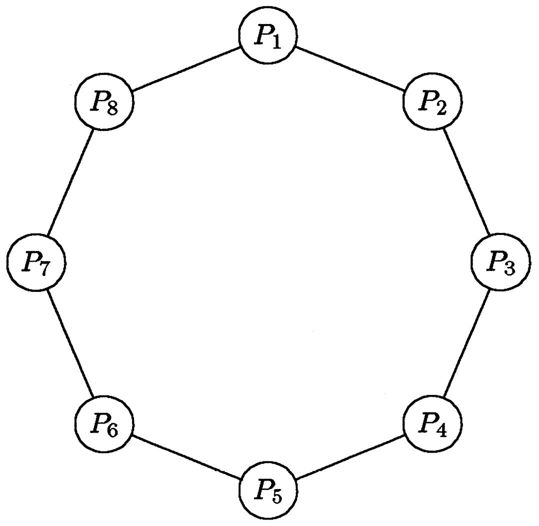

Figure 2: A Processor Ring Consisting of Eight Processors.

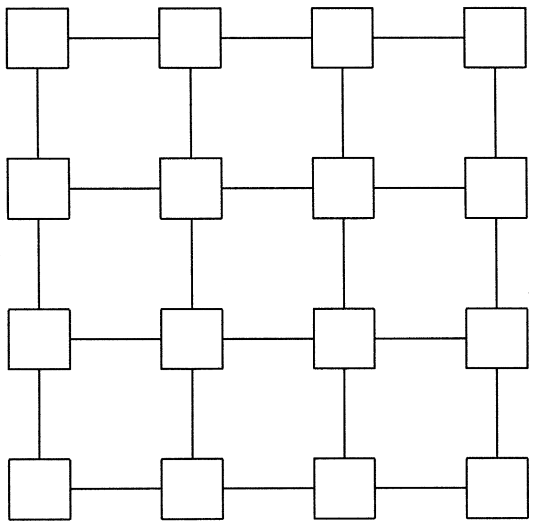

Figure 3: A 4×4 Multiprocessor Grid.

are considered to work in lock step where one step corresponds to the computation time of the slowest processor, which, in particular, implies that identical tasks will take an equal amount of time if started simultaneously on different processors. This assumption is by no means restrictive and its sole purpose is to simplify the complexity analysis and its results. As a matter of fact, most of the parallel algorithms proposed so far can be viewed as SIMD methods, in the sense that a typical parallel loop comprises k *identical* tasks to be executed in parallel.

It is further assumed that communication and arithmetic are not overlapped, which is the case, for instance, when processors are not equipped with I/O co-processors (ie, processors solely

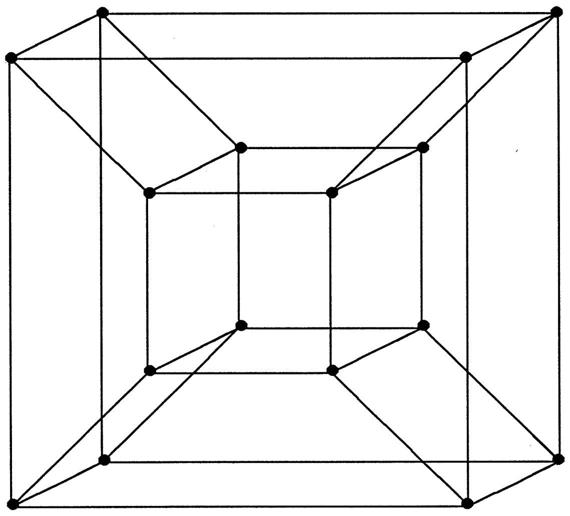

Figure 4: A Hypercube of Dimension 4.

devoted to performing input and output). Yet even when this is not true, it is important to have a realistic measure of what exactly constitutes communication and what computation time, in order to judge the efficiency of an algorithm. We define communication (or data transfer) time as the time to execute an algorithm (in lock step mode) under the assumption that arithmetic can be done in zero time (that is, the arithmetic unit is regarded to be infinitely fast). Arithmetic time can then be defined analogously. The corresponding computation time is at most double of the one resulting from overlapped computation and communication [61].

If processor interconnections are capable of transmitting R words per second, then the inverse is denoted by τ. In general, each transfer of a data *packet* is associated with a *constant* start-up (set-up) time of β, which is independent of the size (the number of words) per packet. Often, the start-up times are (much) larger than the elemental transfer times, that is, $\beta \gg \tau$. The time to send a packet of size N from a processor to its neighbour is $t_T = \beta + N\tau$. On a single processor, a linear combination of two vectors of length N takes time $t_A = \gamma + N\omega$, where γ is the pipe fill time (it is zero for non-pipelined machines), ω the time for one scalar operation and $\gamma \geq \omega$ (again, the start-up time dominates the elemental operation time). For any algorithm the sum of its transfer and arithmetic time, $t_T + t_A$, is simply called its *computation time*.

The following section gives a short overview (with no claims of being complete) over possible implementations of basic Linear Algebra operations on the three loosely coupled architectures. The efficiency of parallel eigenvalue methods depends crucially on the implementation of data transfers, matrix transposition, matrix vector multiplication, and linear system solution.

3.1. Algorithms for the Processor Ring

A multiprocessor ring is one of the simplest interconnection schemes and yet it is one of the most cost-effective architectures when it comes to bridging the gap between future super computers and current vector computers. As suggested in [77] a small number of inexpensive off-the-shelf standard array processors can easily be connected in a ring yielding a machine with the computing power of a CRAY-1. As mentioned before, rings can be emulated without difficulty by most loosely coupled architectures.

Time complexities of elementary data transfers and dense matrix vector multiplication on a processor ring are discussed in [43]. These operations are used to implement various algorithms for solution of dense linear systems by Gaussian elimination on a processor ring. Three ways of assigning matrix elements to particular processors are considered : by rows, by columns and by diagonals. A summary of the results obtained follows.

The communication times are low order terms compared to the arithmetic operation times when the number of processors k is small compared to the order of the matrix N. Both the arithmetic times and the communication times of the triangular system solution methods are low order terms in comparison with those of Gaussian elimination. Allocating non-adjacent rows or columns of the matrix to a processor results in better arithmetic performance but worse communication performance. Allocation of non-adjacent diagonals to a processor results in poor overall performance. The overhead of pivoting is small compared with the cost of Gaussian Elimination; it is however of the same order of magnitude as that of triangular system solution. Pivoting is less expensive for the row-oriented schemes. In [61] lower bounds on the communication complexity for dense Gaussian elimination on bus and ring oriented architecture are shown : the communication time is of order at least $O(N^2)$, independent of the number of processors.

Gaussian elimination for dense systems on a multiprocessor ring is discussed in [71]. Lawrie and Sameh [53] present a technique for solving symmetric positive-definite banded systems, which is a generalisation of a method for tridiagonal system solution on multiprocessors; it takes advantage of different alignment networks for allocating data to the memories of particular processors.

Implementation of the Cholesky factorisation on a ring is discussed in [34]. New algorithms and implementations for the solution of symmetric positive-definite systems are given in [16], as well as minimal time implementations for Toeplitz matrices on linear systolic arrays, which can be easily adapted to loosely-coupled systems.

The literature on solution of linear systems arising from partial differential equations is extensive, the reader is referred to the survey by Ortega and Voight [60]. Iterative methods on rings or linear arrays have been considered by Saad and Sameh [62], Saad, Sameh, and Saylor [63] and more recently by Johnsson, Saad and Schultz [46, 68]. In [68] it was shown that a speed-up of up to $O(\sqrt{N})$ can be achieved when the system arises from an elliptic partial differential equation. Although implementations of direct sparse matrix techniques on vector and parallel computers have been considered by Duff [21], there is very little work dealing with parallel implementations of sparse direct solutions; parallel nested dissection for finite element problems is discussed in [28].

3.2. Algorithms for the Two-Dimensional Processor Grid

A lower bound for the complexity of communication in dense Gaussian Elimination on a two-dimensional grid of processors is $O(N^2/\sqrt{k}) + O(N\sqrt{k})$ [61] when no overlapping of successive steps in Gaussian elimination takes place and $O(N^2/\sqrt{k}) + O(\sqrt{k})$ for pipelined algorithms. In the spirit of the 'wavefront concept' made popular by S.Y. Kung [50] data flow algorithms for dense Cholesky factorisation are developed in [59]. Six different implementations of the dense Cholesky factorisation, depending on the arrangement of the three loop indices, on a processor grid are discussed and compared in [26, 34]. The preferred variant turns out to be a computation of the Cholesky factor by columns, whereby previously computed columns are accessed columnwise.

The idea for transposing dense and banded matrices on two-dimensional architectures was first developed for systolic arrays in [13, 40] and carries over right away to multiprocessor systems; similar ideas can be found in [58].

3.3. Algorithms for the Hypercube

The hypercube topology has been the focus of much recent research in parallel computation; since it can easily emulate many other architectures, the first task is the assignment of data to the processors so as to optimise processor utilisation. Saad and Schultz [65] establish general properties of the hypercube, while Bhatt and Ipsen [6] present algorithms for efficient embeddings of trees onto hypercubes for potential employment in adaptive numerical computations. Chan and

Saad [14] propose a mapping of the grid points onto the hypercube so as to minimise processor communication in Multigrid methods.

Saad and Schultz [66] discuss the problem of solving banded linear systems on ring, mesh and hypercube architectures. It is concluded that the concept of *one best algorithm* for a given architecture is no longer valid. For instance, consider a simple banded linear system of half-bandwidth ν and order N. It is not realistic to assume, as often done, that the half-bandwidth ν matches exactly the number of processors k, ie, that $\nu = k$ or that $\nu^2 = k$. In reality the total number of available processors k is fixed, and one has to determine the best way of solving the system for different values of ν and N. In the extreme case of a tridiagonal matrix, where $\nu = 1$, Gaussian elimination is not parallelisable, and therefore should be excluded, while the cyclic reduction algorithm [44] which is not advantageous for sequential machines is highly parallel and should be selected. At the other extreme, when the half-bandwidth ν is very large with respect to k, simple banded Gaussian elimination with the rows of the matrix uniformly distributed over the processors performs best [66]. Similar observations can be made for ADI methods [46, 64]. Thus, it appears that in the standard software packages of tomorrow one will find several different codes for solving the same problem : according to parameters like size of the problem, number of floating point operations per second and communication bandwidth the program would *dynamically* choose the best alternative.

4. Eigenvalue Algorithms

So far, most of the work on parallel computation of eigenvalues for (symmetric) matrices has seemed to concentrate on the development of systolic array algorithms and hardware – mainly with signal processing applications in mind.

4.1. Methods for (Small) Dense Matrices

There is a variety of systolic array implementations for the symmetric eigenvalue problem based on the shifted QR algorithm with Givens' rotations (the exception is [45] which makes use of Householder transformations). One can use either the arrays for orthogonal factorisations [1, 7, 31, 36, 37, 54] (see [5, 41] for the implementation of 'Fast' rotations) or the ones constructed specifically for the solution of the tridiagonal [36, 57], positive-definite tridiagonal [27] or banded [75] eigenvalue problem; however no satisfactory way for an efficient shift computation has been found. Other designs for symmetric tridiagonal eigenvalue computations include a doubling version of the QR method [3], methods based on isospectral flows [2], and Newton's method or bisection [75]. Often a second, different set of arrays is required to reduce by similarity transformations the original matrix to tridiagonal or banded form [36, 74, 75]. Earlier papers for the solution of the tridiagonal eigenvalue problem on more general parallel architectures suggest the use of multiple Sturm sequences and bisection [49], and computation of the QR iteration via recurrence equations [72].

Parallel implementations of Jacobi's method to compute the eigenvalues of dense symmetric matrices have been considered in [49, 52] as early as 1971. Implementations for the ICL DAP [22] can be found in [56] and improved versions for systolic array implementations [9] seem to be most promising in terms of actual physical realisation since no shift computations are necessary and the architectures are simple; a modified algorithm is presented in [76] to deal with problems whose size does not match the number of available processors.

A systolic array that solves the generalized eigenvalue problem for dense matrices via the QZ algorithm is discussed in [8]. Suggestions for parallel computation of certain instances of the generalized and the nonsymmetric eigenvalue problem, as well as for Lanczos method, are presented in [70].

The divide-and-conquer approach to the tridiagonal eigenvalue problem introduced by Cuppen [15] has been advocated for implementations on tree machines [48], shared-memory architectures [20] and the hypercube [42].

In signal processing applications, the preferred approach is to design arrays that compute the singular values of the Cholesky factors of the covariance matrix directly instead of employing arrays for computing the eigenvalues of the symmetric positive-definite covariance matrix itself. Examples include arrays based on the Jacobi method [10, 11, 55] and on Givens' rotations [36, 39, 76]. The iterative reduction to triangular form of a square matrix by Schur rotations is suggested in [59] for an architecture similar to the the systolic array in [55].

For general two-dimensional processor grids, a dataflow algorithm similar to the one for Cholesky factorisation is developed [59] for congruence transformations.

4.2. Methods for Large Sparse Matrices

Generally speaking large sparse eigenvalue problems have rarely been examined from the parallel point of view, since the parallel algorithms are either trivial extensions of those for solving linear systems or obvious adaptations of sequential algorithms. For instance, a good shift-and-invert technique can be implemented for Lanczos algorithm if the inner loops, which consist of system solutions, can be efficiently parallelised. As a second example, the Subspace Iteration method is a perfectly parallelisable method and might actually constitute a reasonable choice in multiprocessor environments despite its sluggishness on sequential machines. Although actual comparisons between parallel implementations of this technique and Lanczos method in the symmetric case are in order, we still expect Lanczos algorithm to be superior. Along different lines Sameh and Wisniewski [73] and Wisniewski [81] propose an approach based on trace minimisation for solving the generalised eigenvalue problem $Ax = \lambda Bx$, where the trace of $W^T AW$ is minimised over all $N \times M$ matrices W that are B-orthonormal.

An interesting idea by Grimes et al. [33] is the use of Lanczos algorithm, the preferred method for solution of *large sparse* symmetric eigenvalue problems, to solve even relatively *small and dense* problems. The tests conducted on a Cray-XMP/24 show that Lanczos algorithm performs better on dense matrices of order 219 to 1496 than the Cray-optimised EISPACK routines. Thus, vector and parallel machines may result in unexpected changes regarding the range of applicability of classical algorithms.

Large sparse nonsymmetric eigenvalue problems are crucial in the analysis of complex dynamic systems. Due to nonlinearities, the numerical nature of these problems is so intractable that scientists and engineers often abandon them and resort to simplified models. In this situation the use of massive parallel computing could be a decisive factor as it might enable the solution of presently intractable problems. Nonsymmetric eigenvalue methods will benefit very strongly from an increase in computational power and parallelism. It is hoped that these difficult problems will be tackled once reasonably reliable and user-friendly parallel machines appear on the market.

5. Outlook

For multiprocessor systems with regular communication topologies, such as the ones discussed here, novel techniques promise 'automatic' design of many algorithms. Progress in this area has been made especially for the implementation of algorithms on systolic arrays (surveys can be found in [18, 24]). These approaches are easily adaptable to more general multiprocessor architectures with regular communication topologies. Given sets of recurrence equations (eg, a FORTRAN program consisting of several sets of nested loops) the design methodology developed in [17, 18] delivers the description of provably optimal, regular, parallel architectures for their implementation. Upon specification of certain constraints for the resulting architecture, eg, avoidance of broadcasting or restriction to nearest-neighbour communication, the methodology generates an optimal mapping onto a given architecture. Application of this design methodology will allow the numerical analyst to concentrate on the algorithm development, particularly on modifications for improved numerical behaviour or pipelinability, rather than on the mapping or implementation process.

References

[1] Ahmed, H.M., Delosme, J.-M. and Morf, M., *Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing,* IEEE Computer, 15 (1982), pp. 65–82.

[2] Ang, P.H., Delosme, J.-M. and Morf, M., Concurrent Implementation of Matrix Eigenvalue Decomposition based on Isospectral Flows, *Proc. 27th Ann. Symp. of SPIE,* 1983.

[3] Ang, P.H. and Morf, M., Concurrent Array Processor for Fast Eigenvalue Computations, *Proc. 1940 IEEE ICASSP,* 1984, pp. 34/A.2.1–4.

[4] Babbage. H.P., *Babbage's Analytical Engine,* Mon. Not . Roy. Astron. Soc., 70 (1910), pp. 517–26.

[5] Barlow, J.L. and Ipsen, I.C.F, *Scaled Givens Rotations for the Solution of Linear Least Squares Problems on Systolic Arrays,* SIAM J. Sci. Stat. Comp., (1986).

[6] Bhatt, S.N. and Ipsen, I.C.F., *Embedding Trees in the Hypercube,* Research Report 443, Dept Computer Science, Yale University, 1985. Submitted for publication.

[7] Bojanczyk, A., Brent, R.P. and Kung, H.T., *Numerically Stable Solution of Dense Systems of Linear Equations Using Mesh-Connected Processors,* SIAM J. Sci. Stat. Comp., 5 (1984), pp. 95–104.

[8] Boley, D., *A Parallel Method for the Generalized Eigenvalue Problem,* Technical Report 84-21, Dept Computer Science, University of Minnesota, 1984.

[9] Brent, R.P., and Luk, F.T.,, .*The Solution of Singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays,* SIAM J. Sci. Stat. Comput., 6 (1985), pp. 69–84.

[10] Brent, R.P., Luk, F.T., and van Loan, C.F., *Computation of the Singular Value Decomposition Using Mesh-Connected Processors,* J. VLSI Computer Systems, 1 (1984). To appear.

[11] ————, Computation of the Generalized Singular Value Decomposition Using Mesh-Connected Processors, *Proc. SPIE Symp. 431 (Real Time Signal Processing VI),* 1983, pp. 66–71.

[12] Browning, S.A., *The Tree Machine : A Highly Concurrent Computing Environment,* Technical Report TR-3760, Dept Computer Science, California Institute of Technology, 1980.

[13] Cappello, P.R. and Steiglitz, K., Selecting Systolic Designs Using Linear Transformations of Space-Time, *Proc. SPIE Symp. 549 (Real Time Signal Processing VII),* 1984, pp. 75–85.

[14] Chan, T.F. and Saad, Y., *Multigrid Algorithms on the Hypercube Multiprocessor,* Research Report 368, Dept Computer Science, Yale University, 1985.

[15] Cuppen, J.J.M., *A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem,* Num. Math., 36 (1981), pp. 318–40.

[16] Delosme, J.-M. and Ipsen, I.C.F., *Parallel Solution of Symmetric Positive Definite Systems with Hyperbolic Rotations,* Lin. Alg. Appl., (1986).

[17] ————, *Efficient Systolic Arrays for the Solution of Toeplitz Systems : An Illustration of a Methodology for the Construction of Systolic Architectures in VLSI,* Research Report 370, Dept. of Computer Science, Yale University, 1985.

[18] ————, An Illustration of a Methodology for the Construction of Efficient Systolic Architectures in VLSI, *Proc. Second Int. Symposium on VLSI Technology, Systems and Applications,* Taipei, Taiwan, 1985, pp. 268–73.

[19] Dongarra, J.J. and Duff, I.S., *Advanced Architecture Computers,* Technical Memorandum 57, Argonne National Laboratory, 1985.

[20] Dongarra, J.J. and Sorensen, D.C., A Fast Algorithm for the Symmetric Eigenvalue Problem, *IEEE ARITH7 Conference,* University of Illinois, Urbana, 1985.

[21] Duff, I.S., *Parallel Implementation of Multifrontal Techniques,* Technical Memorandum 49, Argonne National Laboratory, 1985.

[22] Flanders, P.M., Hunt, D.J., Reddaway, S.F. and Parkinson, D., Efficient High Speed Computing with the Distributed Array Processor, *High Speed Computer and Algorithm Organization,* Academic Press, 1977, pp. 113–28.

[23] Flynn, M.J., *Some Computer Organizations and their Effectiveness,* IEEE Trans. Comp., C-21 (1972), pp. 948–60.

[24] Fortes, J.A.B., Fu, K.S. and Wah, B.W., Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays, *Proc. ICASSP,* 1985, pp. 8.9.1–4.

[25] Frailong, J.M, Jalby, W. and Lenfant, J., XOR Schemes: A Flexible Data Organization in Parallel Memories., *Proc. 1985 Int. Conf. Parallel Processing,* IEEE, 1985, pp. 276–83.

[26] Funderlic, R.E. and Geist, G.A., *Torus Data Flow for Parallel Computation of Missized Matrix Problems,* Technical Report ORNL-6125, Oak Ridge National Laboratory, 1985.

[27] Gajski, D.D, Sameh, A.H. and Wisniewski, J.A., Iterative Algorithms for Tridiagonal Matrices on a WSI-Multiprocessor, *Proc. Int. Conf. Parallel Processing,* IEEE, 1982, pp. 82–9.

[28] Gannon, D., A Note on Pipelining a Mesh Connected Multiprocessor for Finite Element Problems by Nested Dissection, *Proc. 1980 Int. Conf. Parallel Processing,* IEEE, 1980, pp. 197–204.

[29] Gannon, D. and van Rosendale, J., *On the Impact of Communication Complexity in the Design of Parallel Algorithms,* Technical Report 84-41, ICASE, 1984.

[30] Gentleman, W.M, *Some Complexity Results for Matrix Computation on Parallel Processors,* JACM., 25 (1978), pp. 112–115.

[31] Gentleman, W.M., and Kung, H.T., On Stable Parallel Linear System Solvers, *Proc. SPIE (Real Time Signal Processing IV),* 1981, pp. 19–26.

[32] Gottlieb, A., Grishman, R., Kruskal, C.P., McAuliffe, K.P., Rudolph, L. and Snir, M., *The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer,* IEEE Trans. Comp., C-32 (1983), pp. 175–89.

[33] Grimes, R.G., Lewis, J.G., Simon, H., Krakauer, H. and Wei, S.H., *The Solution of Large Dense Generalized Eigenvalue Problems on the Cray X-MP/24,* 1985. Presentation given at the Second SIAM Conf. Par. Proc. and Sci. Comp., Norfolk, VA.

[34] Heath, M.T., *Parallel Cholesky Factorisation in Message-Passing Multiprocessor Environments,* Technical Report ORNL-6150, Oak Ridge National Laboratory, 1985. Submitted to *Parallel Computing.*

[35] Heller, D.E., *A Survey of Parallel Algorithms in Numerical Linear Algebra,* SIAM Review, 20 (1978), pp. 740–777.

[36] Heller, D.E. and Ipsen, I.C.F., Systolic Networks for Orthogonal Equivalence Transformations and their Applications, Penfield, P. ed., *Proc. Conf. Advanced Research in VLSI, 1982,* Artech House, Inc., 1982, pp. 113–22.

[37] Heller, D.E. and Ipsen, I.C.F, *Systolic Networks for Orthogonal Decompositions,* SIAM J. Sci. Stat. Comp., 4 (1983), pp. 261–9.

[38] Hockney, R.W. and Jesshope, C.R., *Parallel Computers,* Adam Hilger Ltd, Bristol, 1981.

[39] Ipsen, I.C.F., Singular Value Decomposition with Systolic Arrays, *Proc. SPIE Symp. 549 (Real Time Signal Processing VII),* 1984, pp. 13–21.

[40] ————, *Stable Matrix Computations in VLSI,* Ph.D. Thesis, The Pennsylvania State University, 1983.

[41] ———, *A Parallel QR Method Using Fast Givens' Rotations,* Research Report 299, Dept Computer Science, Yale University, 1984.

[42] Ipsen, I.C.F. and Jessup, E., *Solving the Symmetric Tridiagonal Eigenvalue Problem on the Hypercube,* Research Report in Preparation, Dept Computer Science, Yale University,

[43] Ipsen, I.C.F., Saad,Y. and Schultz, M.H., *Complexity of Dense Linear System Solution on a Multiprocessor Ring,* Lin. Alg. Appl., (1986).

[44] Johnsson, S.L., *Odd-Even Cyclic Reduction on Ensemble Architectures.,* SIAM J. Sci. Stat. Comp, (1986).

[45] ———, A Computational Array for the QR-Method, Penfield, P. ed., *Proc. Conference on Advanced Research in VLSI, 1982,* Artech House, Inc., 1982, pp. 123–9.

[46] Johnsson, S.L., Saad, Y. and Schultz, M.H., *The Alternating Direction Algorithm on Multiprocessors,* Research Report 382, Dept Computer Science, Yale University, 1985.

[47] Jordan, H.F., A Special Purpose Architecture for Finite Element Analysis, *Proc. Int. Conf. Parallel Processing,* 1978, pp. 263–6.

[48] Krishnakumar, A.S. and Morf, M., A Tree Architecture for the Symmetric Eigenproblem, *Proc. 27th Annual Symp. of SPIE,* 1983.

[49] Kuck, D.J. and Sameh, A.H., Parallel Computation of Eigenvalues of Real Matrices, *Proc. IFIP Congress 1971,* North Holland, Amsterdam, 1972, pp. 1266–72.

[50] Kung, S.-Y., Arun, K.S., Gal-Ezer, R.J. and Bhaskar Rao, D.V, *Wavefront Array Processor : Language, Architecture, and Applications,* IEEE Trans. Comp., C-31 (1982), pp. 1054–66.

[51] Kung, H.T. and Leiserson, C.E., Systolic Arrays (for VLSI), *Sparse Matrix Proceedings,* SIAM, Philadelphia, PA, 1978, pp. 256–82.

[52] Lawrie,D. and Sameh, A.H., *On Jacobi and Jacobi-like Algorithms for a Parallel Computer,* Math. Comput., 25 (1971), pp. 579–90.

[53] Lawrie, D. and Sameh, A.H., *The Computation and Communication Complexity of a Parallel Banded Linear System Solver,* ACM TOMS, 10 (1984), pp. 185–95.

[54] Luk, F.T., *A Jacobi-Like Algorithm for Computing the QR-Decomposition,* Technical Report TR-84-612, Dept Computer Science, Cornell University, 1984.

[55] ———, *A Triangular Processor Array for Computing the Singular Value Decomposition,* Technical Report TR-84-625, Dept Computer Science, Cornell University, 1984.

[56] Modi, J.J. and Pryce, J.D., *Efficient Implementation of Jacobi's Method on the DAP,* Technical Report CUED/f-CAMS/TR.238, Dept Engineering, Cambridge University, 1982.

[57] Moldovan, D.I., Wu, C.I. and Fortes, J.A.B, Mapping an Arbitrarily Large QR Algorithm into a Fixed Size Array, *Proc. 1984 Conf. Parallel Processing,* 1984, pp. 365–73.

[58] O'Leary, D.P., *Systolic Arrays for Matrix Transpose and other Reorderings,* Technical Report 1481, Dept Computer Science, University of Maryland, 1985.

[59] O'Leary, D.P and Stewart, G.W., *Data-Flow Algorithms for Parallel Matrix Computations,* Technical Report 1366, Dept Computer Science, University of Maryland, 1984.

[60] Ortega, J.M. and Voigt, R.G., *Solution of Partial Differential Equations on Vector and Parallel Computers,* SIAM Review, 27 (1985), pp. 149–240.

[61] Saad, Y., *Communication Complexity of the Gaussian Elimination Algorithm on Multiprocessors,* Research Report 348, Dept Computer Science, Yale University, 1985.

[62] Saad, Y. and Sameh, A.H., A parallel Block Stiefel Method for Solving Positive Definite Systems, Schultz, M.H. ed., *Proc. Elliptic Problem Solver Conf.,* Academic Press, 1980, pp. 405–12.

[63] Saad, Y., Sameh, A.H. and Saylor, P., *Solving Elliptic Difference Equations on a Linear Array of Processors,* SIAM J. Sci. Stat. Comp., 6 (1985), pp. 1049–63.

[64] Saad, Y. and Schultz, M.H., *Alternating Direction Methods on Multiprocessors : An Extended Abstract,* Research Report 381, Dept Computer Science, Yale University, 1985.

[65] ————, *Topological Properties of Hypercubes,* Research Report 389, Dept Computer Science, Yale University, 1985.

[66] ————, *Direct Parallel Methods for Solving Banded Linear Systems,* Research Report 387, Dept Computer Science, Yale University, 1985.

[67] ————, *Data Communication in Hypercubes,* Research Report 428, Dept Computer Science, Yale University, 1985.

[68] ————, *Parallel Implementations of Preconditioned Conjugate Gradient Methods,* Research Report 425, Dept Computer Science, Yale University, 1985.

[69] Sameh, A.H., Numerical Parallel Algorithms - A Survey, Lawrie, D. and Sameh, A.H. ed., *High Speed Computer and Algorithm Organization,* Academic Press, 1977, pp. 207–28.

[70] ————, An Overview of Parallel Algorithms in Numerical Linear Algebra, *Proc. 1er Colloque International sur les Methodes Vectorielles et Paraleles en Calcul Scientifique, Bulletin de la Direction des Etudes et Recherches, Serie C,* Electricite De France, 1983, pp. 129–34.

[71] ————, *On Some Parallel Algorithms on a Ring of Processors,* Technical Report , Dept Computer Science, University of Illinois at Urbana-Champaign, 1984.

[72] Sameh, A.H. and Kuck, D.J., *A Parallel QR Algorithm for Symmetric Tridiagonal Matrices,* IEEE Trans. Comp., C-26 (1977), pp. 147–53.

[73] Sameh, A.H. and Wisniewski, J.A., *A Trace Minimization Algorithm for the Generalized Eigenvalue Problem,* SIAM J. Num. Anal., 19 (1982), pp. 1243–59.

[74] Schreiber, R., Systolic Arrays for Eigenvalue Computation, *Proc. SPIE 341 (Real Time Signal Processing V),* 1982, pp. 27–34.

[75] ————, *On the Systolic Arrays of Brent, Luk and Van Loan for the Symmetric Eigenvalue and Singular Value Problems,* Technical Report TRITA-NA-8311, Dept Numerical Analysis and Computer Science, Royal Institute of Technology, Sweden, 1983.

[76] ————, A Systolic Architecture for the Singular Value Decomposition, *Proc. 1er Colloque International sur les Methodes Vectorielles et Paraleles en Calcul Scientifique, Bulletin de la Direction des Etudes et Recherches, Serie C,* Electricite de France, 1983.

[77] Schultz, M.H., *Multiple Array Processors for Ocean Acoustic Problems,* Research Report 363, Dept Computer Science, Yale University, 1985.

[78] Seitz, C.L., *The Cosmic Cube,* CACM, 28 (1985), pp. 22–33.

[79] Shapiro, H.D., *Theoretical Limitations on the Efficient Use of Parallel Memories,* IEEE Trans. Comp., C-27 (1978), pp. 421–28.

[80] Slotnick, D.L, Unconventional Systems, *AFIPS Conf. Proc. 30,* 1967, pp. 477–81.

[81] Wisniewski, J.A., *A Parallel Algorithm for Solving $Ax = \lambda Bx$,* Ph.D. Thesis, University of Illinois at Urbana Champaign, 1980.