Computers Math. Applic. Vol. 25, No. 9, pp. 91–96, 1993
Printed in Great Britain.

0097-4943/93 $6.00 + 0.00

SOLVING THE SYMMETRIC TRIDIAGONAL EIGENVALUE PROBLEM ON HYPERCUBES

KUO-LIANG CHUNG
Department of Information Management
National Taiwan Institute of Technology, Taipei, Taiwan 10772, R.O.C.
WEN-MING YAN
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan 10764, R.O.C.

(Received January 1992; revised and accepted July 1992)

Abstract—Using the methods of bisection and inverse iteration respectively, this paper presents a parallel solver for the calculation of the eigenvalues of a real symmetric tridiagonal matrix on hypercube networks in $O(m_1 \log n)$ time using $\Theta(n^2/\log n)$ processors, where m_1 is the number of iterations. The corresponding eigenvectors problem can be solved in $O(\log n)$ time on the same networks.

1. INTRODUCTION

Given a real symmetric tridiagonal matrix of order $n-1$, $A = (b_{i-1}a_ib_i)$ for $1 \leq i \leq n-1$. Let A_j be the leading $j \times j$ principal submatrix of A and define the characteristic polynomial $Q_j(\lambda) = \det(A_j - \lambda I)$, $1 \leq j \leq n-1$. We assume that no off-diagonal element is zero, i.e., $b_i \neq 0$ for all i, $1 \leq i \leq n-2$. The original problem, namely, solving the symmetric tridiagonal eigenvalues and eigenvectors problem, can be divided into two subproblems if some b_i is equal to zero. By simple determinantal expansion [1], the Sturm sequence is derived by

$$Q_j(\lambda) = (a_j - \lambda)Q_{j-1}(\lambda) - b_{j-1}^2 Q_{j-2}(\lambda) \tag{1}$$

with the initial conditions $Q_0(\lambda) = 1$ and $Q_1(\lambda) = a_1 - \lambda$.

Based on modified Sturm sequence evaluations coupled with the inverse iteration [2,3], where (1) is rescaled and replaced by the nonlinear recurrence: $P_1(\lambda) = a_1 - \lambda$, $P_i = a_i - \lambda - (b_{i-1}^2)/(P_{i-1})$, Ipsen and Jessup [4] proposed a parallel algorithm consisting of six complicated parts to solve the eigenvalues and eigenvectors problems on a p-processor hypercube. In their algorithm, each processor in the hypercube network computes k $(= n/p)$ eigenvalues and k eigenvectors of A. Without the time-complexity analysis, they presented the numerical results and timings for Intel's iPSC-1. Evans and Margaritis [5] proposed a pipelining systolic array to find all the eigenvalues in $O(mn)$ time using $\Theta(n)$ processors, where m is the number of iterations. In addition, by utilizing the LU decomposition method with partial pivoting, the corresponding eigenvectors can be solved in $O(n^2)$ time using $\Theta(1)$ processors.

By using the methods of bisection and inverse iteration respectively, this paper presents a parallel solver for the calculation of the eigenvalues of a real symmetric tridiagonal matrix on hypercube networks in $O(m_1 \log n)$ time using $\Theta(n^2/\log n)$ processors, where m_1 is the number of iterations. The corresponding eigenvectors problem can be solved in $O(\log n)$ time on the same networks. Under the same cost, theoretically, our solver is faster than the one by Evans and Margaritis.

This research is supported by the National Science Council of the Republic of China under contract NSC80-0415-E011-10. We are indebted to the reviewers for making some valuable suggestions and corrections that lead to the improved version of the paper.

2. THE COMPUTATION OF THE EIGENVALUES

To use the Givens' bisection method for finding all the eigenvalues, we first require the following theorem.

THEOREM 2.1. *[1] The number of eigenvalues smaller than a given x is equal to the number of sign disagreements, $S(x)$, in the Sturm sequence $Q_0(x), Q_1(x), \ldots, Q_{n-1}(x)$.*

In order that the above theorem should be meaningful for all values of x, we must associate a sign with zero values of $Q_i(x)$. If $Q_i(x) = 0$, then $Q_i(x)$ is taken to have the same sign to that of $Q_{i-1}(x)$. It is easily proved that no two consecutive terms $Q_{i-1}(x)$ and $Q_i(x)$ are zero [3].

For finding the feasible interval, say, (α_0, β_0), to confine every eigenvalue of A, we need the following theorem due to Gerschgorin [3].

THEOREM 2.2. *Every eigenvalue of A lies in the interval (α_0, β_0), where $\alpha_0 = \min_{1 \le i \le n-1} (a_i - |b_{i-1}| - |b_i|)$ and $\beta_0 = \max_{1 \le i \le n-1} (a_i + |b_{i-1}| + |b_i|)$.*

Let $\lambda_i(A)$ denote the i^{th} largest eigenvalue of A. From Theorem 2.2, we know that any $\lambda_i(A)$ must lie in the interval (α_0, β_0). First, let $c_1 = (\alpha_0 + \beta_0)/2$ be the middle point of (α_0, β_0). Then compute the sequence $Q_0(c_1), Q_1(c_1), \ldots, Q_{n-1}(c_1)$ and hence determine $S(c_1)$. If $S(c_1) < i$, then set $\alpha_1 = c_1$ and $\beta_1 = \beta_0$; otherwise set $\alpha_1 = \alpha_0$ and $\beta_1 = c_1$. Next, let $c_1 = (\alpha_1 + \beta_1)/2$, and we repeat the above bisection process again. Finally, we can locate $\lambda_i(A)$ in an interval (a_{m_1}, b_{m_1}) of width $(\beta_0 - \alpha_0)/2^{m_1}$ after m_1 iterations. Totally, there are $n-1$ such eigensystems to be solved.

For each task to compute $\lambda_i(A)$, observe that the major work in each iteration is to first compute the prefix values of $Q_{n-1}(c_1)$, and then to determine the sign disagreement $S(c_1)$. Using the recursive-doubling method [6], the Sturm sequence (1) can be rewritten as

$$Q'_j(c_1) = M_j(c_1)\, Q'_{j-1}(c_1),$$

where $Q'_j(c_1) = (Q_j(c_1), Q_{j-1}(c_1))^t$ with $Q'_0 = (1,0)^t$ and $M_j(c_1) = (v, w)$, where $v = (a_j - c_1, 1)^t$, $w = (-b_{j-1}^2, 0)^t$ and $b_0 = 0$. After finishing the prefix computation of $Q'_{n-1}(c_1)$, each $Q_i(c_1)$ can be determined directly when we select the first component of $Q'_i(c_1)$.

An overlaid tree network to perform the typical prefix computation has been presented in [7]. Given $Q'_0, M_1(c_1), M_2(c_1), \ldots, M_7(c_1)$, an example of how the overlaid tree network works for the prefix computation of $Q_7(c_1)$ is shown in Figure 1.

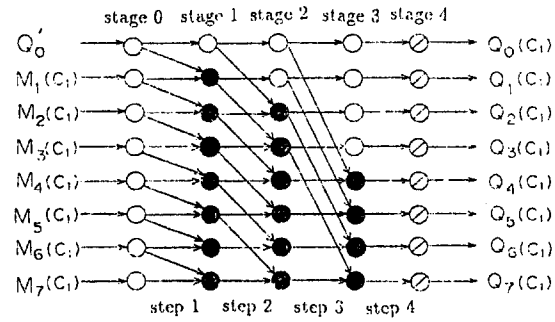


Figure 1. A network for computing the prefix values of $Q_7(c_1)$.

In Figure 1, the white nodes in the leftmost stage 0 are used for preparing input data only. The white nodes in stages 1, 2, and 3 are used for transmitting data only. The black nodes perform the 2×2 matrix multiplications. The slash nodes $\oslash$ select the desired data. Initially, 8 input data $Q'_0, M_1(c_1), M_2(c_1), \ldots, M_7(c_1)$ are fed into the white nodes in stage 0. Passing through five stages, the prefix values of $Q_7(c_1)$ are obtained. For exposition, let the number of input data Q'_0, $M_1(c_1), M_2(c_1), \ldots, M_{n-1}(c_1)$ be a power of 2, i.e., $n = 2^r$. Therefore, there are $\log n + 2$ stages in this network, and the computation time is only $O(\log n)$ time. If each node is implemented by a processor, the number of processors is as large as $\Theta(n \log n)$, which is reduced to $\Theta(n/\log n)$ in Section 3, while retaining the same time complexity.

3. MAPPING INTO THE HYPERCUBE NETWORK

We now present how to map the prefix computations introduced in Section 2 into a hypercube network. An r-dimensional hypercube [8] has $n = 2^r$ nodes. Each node is incident to r $(= \log n)$ other nodes, one across each dimension. The node can be labeled from 0 to $2^r - 1$ (in binary-reflected Gray code, which will be defined later), and two processors are linked if and only if their labels differ in exactly one bit position. Figure 2 shows the construction of a 16-node hypercube from two 8-node hypercubes.

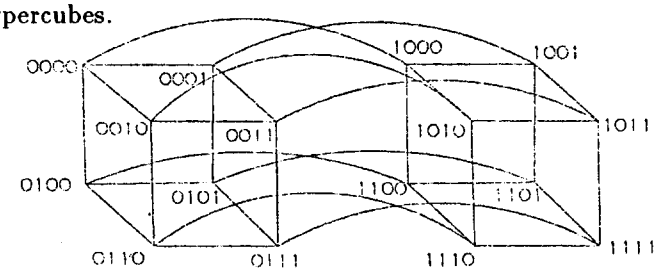


Figure 2. A 16-node hypercube.

Refer to Figure 1 again. Denote the i^{th} processor of stage k by $P(i,k)$, $0 \le i \le n-1$, $1 \le k \le \log n$. At step k (from stage $k-1$ to stage k), $1 \le k \le \log n$, processor $P(i,k-1)$ sends its data to processors $P(i,k)$ and $P(j,k)$, where $j = i + 2^{k-1}$. Processor $P(j,k)$ receives the data sent from $P(j-2^{k-1},k-1)$ and calculates it with the data sent from $P(j,k-1)$.

Let $G_i = g_{\log n-1} g_{\log n-2} \cdots g_0$ be the binary-reflected Gray code of $i = b_{\log n-1} b_{\log n-2} \cdots b_0$, i.e.,

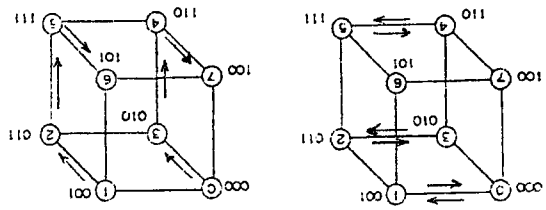
$$g_{\log n-1} = b_{\log n-1},$$
$$g_j = b_{j+1} \oplus b_j \qquad \text{for } 0 \le j \le \log n - 2,$$

where $\oplus$ is the exclusive-or operator [9]. We denote the i^{th} processor by G_i (simply called processor G_i) and will simulate all the functions of $P(i,k)$, $1 \le k \le \log n$, on that processor. By [10], we have the following lemma and theorem.

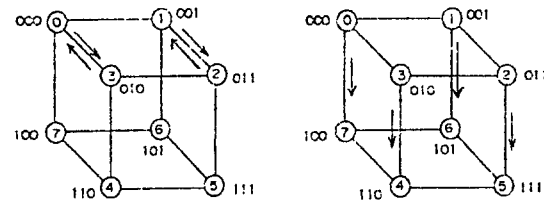
LEMMA 3.1. *The binary-reflected Gray codes G_i and $G_{i+2^{k-1}}$, $k > 1$, differ in precisely two bit positions. Those bits are g_k and g_s, where s is the bit position in which the carry stops propagating when 2^{k-1} is added to i.*

THEOREM 3.2. *In simulating the overlaid tree network on the n-node hypercube network, two data transfer steps in the hypercube are needed for each step in the overlaid tree network except the first, which needs only one data transfer step. Let the edge in the hypercube be bidirectional. The following routing guarantees disjoint data transfer paths: processor G_i sends data to processor x with the address obtained by complementing the lower order bit where G_i and $G_{i+2^{k-1}}$ differ, then processor x sends its data to processor $G_{i+2^{k-1}}$.*

The simulations of Step 2 and Step 3 of Figure 1 on an 8-node cube are illustrated in Figure 3a and 3b respectively. For processor $P(i,1)$ of Figure 1, the corresponding processor in the 8-node cube as shown in Figure 3 is denoted by the index i, which is encapsulated in a circle, and is associated with G_i. First, the data in processor G_i, $0 \le i \le 5$, are transmitted to the intermediate processor x simultaneously. The direction of each data transfer is denoted by the arrow as the left figure of Figure 3a shows. Next, the data in processor x are transmitted to processor G_{i+2} as the right figure of Figure 3a shows, and then the 2×2 matrix multiplications are performed in parallel. For simplicity, throughout the rest of the paper, we assume that the cost of one data transfer is equal to the cost of one computation. This assumption is reasonable since the ratio of one data transfer cost and one computation cost can be bounded by a constant factor [11]. This assumption will not alter the result of complexity analysis when we use the big O notation. As a result, Step 2 of Figure 1 can be simulated by two steps on the 8-node cube. By the same way, Step 3 of Figure 1 can be simulated by two steps as shown in Figure 3b. In general, the prefix values of $Q_{n-1}(c_1)$ can be computed in $O(\log n)$ steps on a hypercube of n nodes. Note that the tree-like computations of Figure 1 can also be mapped onto an Omega network [12], and



(a) Simulation of Step 2.



(b) Simulation of Step 3.

Figure 3. Simulation of Figure 1 on an 8-node hypercube.

if the adjacent pairs of nodes in an Omega network correspond to pairs of nodes along different dimensions of a hypercube, then the above mapping between trees and hypercubes become clear.

Suppose we have t processors, $t \leq n$. Let the n input data $(Q'_0, M_1(c_1), \ldots, M_{n-1}(c_1))$ be evenly divided into t pipes to be stored in the processors separately. Algorithm 3.1 describes how the hypercube network works.

ALGORITHM 3.1.

PHASE 1. (Local computations)

Each processor G_i sequentially computes n/t prefix values from its corresponding pipe of n/t data and stores them in its local memory. The register $R(G_i)$ residing in processor G_i contains the result $M_{(i+1)n/t-1}(c_1) \times \cdots \times M_{in/t}(c_1)$, $0 \leq i \leq t-1$, where $M_0(c_1) = Q'_0$.

PHASE 2. (Global prefix computations)

All the processors work together using the routing mechanism described in Theorem 3.2 for prefix computations on the content of $R(G_i)$'s. The register $R(G_i)$ holds the value $Q'_{(i+1)n/t-1}(c_1)$, $0 \leq i \leq t-1$.

PHASE 3. (Adaptation)

Each processor G_i except processor G_0 receives the value of $R(G_{i-1})$ from processor G_{i-1}, and sequentially modifies the prefix values calculated in Phase 1 by multiplying the value of $R(G_{i-1})$ to each of those local prefix values.

Both Phase 1 and Phase 3 contain $O(n/t)$ computation steps. Phase 2 contains $2\log t - 1$ data transfer steps and $O(\log t)$ computation steps. By the assumption of one data transfer cost being equal to one computation cost, the three-phase algorithm takes $O(n/t + \log t)$ time to finish computing all the prefix values of $Q'_{n-1}(c_1)$. It needs $O(1)$ time to obtain the prefix values of $Q_{n-1}(c_1)$.

We use $n = 8$ and $t = 4$ as an example to illustrate the algorithm briefly. Initially, the 8 input data $(M_0(c_1), M_1(c_1), \ldots, M_7(c_1))$ are evenly divided into 4 pipes, each containing 2 data. The processor G_i, $0 \leq i \leq 3$, has the data $M_{2i}(c_1)$ and $M_{2i+1}(c_1)$. After Phase 1 of Algorithm 3.1, the register $R(G_0)$ has the resultant of $M_1(c_1) \times M_0(c_1)$, $R(G_1)$ has the resultant of $M_3(c_1) \times M_2(c_1)$, and so on. After Phase 2, $R(G_0)$ has the resultant of $M_1(c_1) \times M_0(c_1)$, $R(G_1)$ has the resultant of $M_3(c_1) \times \cdots \times M_0(c_1)$, and so on. After Phase 3 for adaptation, the prefix values of $Q'_7(c_1)$ are obtained, and then the prefix values of $Q_7(c_1)$ are obtained directly. That is, processor G_0 has the values $\{Q_0(c_1), Q_1(c_1)\}$, G_1 has the values $\{Q_2(c_1), Q_3(c_1)\}$, and so on.

For the subsequent parts of computation, namely, computing the number of sign disagreements, $S(c_1)$, the similar three-phase concept works equally well. In Phase 1, each processor G_i does sum up the number, say $S_i(c_1)$, of its corresponding pipe of sign disagreement sequentially. This step

needs about $O(n/t)$ computation steps. In Phase 2, processor G_i sends the value of $Q_{(i+1)n/t-1}$ to processor G_{i+1} for $0 \leq i \leq t-2$, and the value of $S_j(c_1)$, $1 \leq j \leq t-1$, is updated based on the sign disagreement of $Q_{jn/t-1}$ and $Q_{jn/t}$. In Phase 3, these new values of $S_i(c_1)$'s, $0 \leq i \leq t-1$ are summed by a tree method. It takes about $O(\log t)$ steps because it is well known that a tree-structured computation can be performed in logrithmic time on a hypercube network. We have the following lemma.

LEMMA 3.3. *The number of sign disagreements, $S(c_1)$, in the Sturm sequence of* (1) *can be determined in $O(n/t + \log t)$ time on a hypercube network of t processors.*

The performance of a parallel algorithm can be measured by Cost = Number of Processors × Execution Time. Given a problem, if the cost of a parallel algorithm matches the sequential time lower bound within a constant, the parallel algorithm is said to be cost-optimal. In the case of determining the sign disagreement in the Sturm sequence, since there are $n-1$ signs to be produced and calculated, the sequential time lower bound is $\Omega(n)$. For the bootstrapping method [13], if we set $t = \Theta(n/\log n)$ in Lemma 3.3, we have the following theorem.

THEOREM 3.4. *The value of $S(c_1)$ can be cost-optimally determined in $O(\log n)$ time on a hypercube network of $\Theta(n/\log n)$ processors.*

Now let's return to the eigenvalues problem. From Theorem 3.4, one can see that a sign disagreement can be determined in $O(\log n)$ time on a $\log(n/\log n)$-dimensional hypercube. To solve $n-1$ sign disagreements simultaneously, we need $n-1$ such hypercubes. A natural approach is to connect these $n-1$ hypercubes, each of them an $n/\log n$-node hypercube, by a simple network. Therefore, we have the following main result.

THEOREM 3.5. *The eigenvalues problem can be solved in $O(m_1 \log n)$ time on hypercube networks of $\Theta(n^2/\log n)$ processors, where m_1 has been discussed in Section 2.*

4. THE CALCULATION OF THE EIGENVECTORS

Using the inverse iteration method [2,3], the eigenvector $\mathbf{x}$ of A with respect to the eigenvalue λ can be approximated by solving the symmetric tridiagonal system

$$(A - \lambda I)\mathbf{x} = b, \tag{2}$$

where b is an arbitrarily normalized vector.

If we apply the cost-optimal parallel tridiagonal system solver [14], then (2) can be solved by means of Gaussian elimination and backward substitution. The first iterate of the eigenvector $\mathbf{x}$, the approximated eigenvector $\mathbf{x_1}$ can be determined in $O(\log n)$ time on a hypercube using $\Theta(n/\log n)$ processors. Next we solve the system

$$(A - \lambda I)\mathbf{x_2} = \mathbf{x_1}$$

again, and the improved approximated eigenvector $\mathbf{x_2}$ is obtained.

In practice, if we select a suitable b, then the second vector $\mathbf{x_2}$ would usually be a very good approximation to the exact eigenvector corresponding to the specific eigenvalue λ. How to determine the choice of initial vector b is suggested in [3].

For each eigenvalue λ_i, $1 \leq i \leq n-1$, the corresponding approximated eigenvector can be determined in $O(\log n)$ time on a $\log(n/\log n)$-dimensional hypercube. To solve $n-1$ such symmetric tridiagonal systems simultaneously, we need $n-1$ such hypercubes. By the same arguments in Section 3, we have the following theorem.

THEOREM 4.1. *The eigenvectors problem can be solved in $O(\log n)$ time on hypercube networks of $\Theta(n^2/\log n)$ processors.*

5. CONCLUDING REMARKS

A parallel algorithm for solving the symmetric tridiagonal eigenvalues and eigenvectors problem has been presented. On hypercube networks of $\Theta(n^2/\log n)$ processors, the eigenvalues problem can be solved in $O(m_1 \log n)$ time; the corresponding eigenvectors problem can be solved in $O(\log n)$ time.

Under the same cost, theoretically, our parallel solver is faster than the one proposed by Evans and Margaritis [5]. However, our parallel solver may suffer from the possibility of overflow and underflow in the absence of rescaling before each iteration. In practice, if the floating-point number system has high precision, the drawback can be remedied.

REFERENCES

1. J.W. Givens, A method of computing eigenvalues and eigenvectors suggested by classical results on symmetric matrices, *U.S. Nat. Bur. Standards Applied Mathematics Series* **29**, 117–122 (1953).
2. W. Barth, R.S. Martin and J.H. Wilkinson, Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection, *Numer. Math.* **9**, 386–393 (1967).
3. J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, (1965).
4. I.C.F. Ipsen and E.R. Jessup, Solving the symmetric tridiagonal eigenvalue problem on the hypercube, *SIAM J. Sci. Stat. Comput.* **11** (2), 203–229 (1990).
5. D.J. Evans and K. Margaritis, Systolic designs for the calculation of the eigenvalues and eigenvectors of a symmetric tridiagonal matrix, *Inter. J. Computer Math.* **33**, 1–12 (1990).
6. H.S. Stone, An efficient parallel algorithm for the solution of a tridiagonal linear system of equations, *J. ACM* **20** (1), 27–38 (1973).
7. H.S. Stone, *Introduction to Computer Architecture*, Science Research, Chicago, IL, (1980).
8. C.L. Seitz, The cosmic cube, *Comm. ACM* **28** (1), 22–33 (1985).
9. E.M. Reigold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, (1977).
10. S.L. Johnson, Solving tridiagonal systems on ensemble architecture, *SIAM J. Sci. Stat. Comput.* **8** (3), 354–392 (1987).
11. Y. Saad and M.H. Schultz, Data communication in hypercubes, *J. of Parallel and Distributed Computing* **6**, 115–135 (1989).
12. H.S. Stone, Parallel processing with the perfect shuffle, *IEEE Trans. on Computers* **C-20** (2), 153–161 (1971).
13. D.H. Greene and D.E. Knuth, *Mathematics for the Analysis of Algorithms*, 2nd ed., Birkhäuser, Boston, (1982).
14. F.C. Lin and K.L. Chung, A cost-optimal parallel tridiagonal system solver, *Parallel Computing* **15**, 189–199 (1990).