CONF-930331--17

Chapter 14
Parallel Environments and Tools

Optimal Eigenvalue Computation on a Mesh Multiprocessor *

S. Crivelli † E. R. Jessup †

Abstract

In this paper, we compare the costs of computing a single eigenvalue of a symmetric tridiagonal matrix by serial bisection and by parallel multisection on a mesh multiprocessor. We show how the optimal method for computing one eigenvalue depends on such variables as the matrix order and parameters of the multiprocessor used. We present the results of experiments on the 520-processor Intel Touchstone Delta to support our analysis.

1 Introduction

Let $T = [\eta_j, \zeta_j, \eta_{j+1}]$ be an $n \times n$ real symmetric tridiagonal matrix with diagonal elements ζ_j, $j = 0, \ldots, n-1$, and off-diagonal elements $\eta_j \neq 0$, for $j = 1, \ldots, n-1$. The number of eigenvalues of T smaller than a given value λ is equal to the number of negative terms $\sigma(\lambda)$ in the Sturm sequence $\{f_i(\lambda)\}$ defined as

$$\begin{aligned} f_0(\lambda) &= \zeta_0 - \lambda \\ f_i(\lambda) &= \zeta_i - \lambda - \frac{\eta_i^2}{f_{i-1}(\lambda)} \qquad i = 1, \ldots, n-1. \end{aligned} \tag{1}$$

The number of eigenvalues in the interval $(\lambda_{-1}, \lambda_0]$ is the difference $\sigma(\lambda_0) - \sigma(\lambda_{-1})$ [2,11].

An eigenvalue of T can be computed by repeatedly bisecting the initial Gerschgorin interval and determining the number of eigenvalues in each interval half [5]. Empty intervals are discarded from the search area, and the interval containing the eigenvalue is further bisected until the eigenvalue has been confined to an interval with width smaller than a given tolerance. Multisection is a generalization of bisection that recursively splits the initial interval into $p + 1 \geq 2$ subintervals [6].

Although the recurrence formula (1) is intrinsically sequential, it is possible to achieve parallelism by simultaneously evaluating the Sturm sequence at p interior points of the search interval, one evaluation per processor, or by computing different eigenvalues in parallel. Several parallel bisection and multisection procedures have been proposed but little has been proven about their efficiencies. Experiments on shared-memory multiprocessors indicate that a combination of bisection and multisection is more efficient than using bisection alone in parallel [8], while experiments on distributed-memory multiprocessors suggest the opposite [7]. Bernstein and Goldstein [3] note that multisection may create a large number of tasks associated with empty

*Both authors were funded by DOE contract DE-FG02-92ER25122 and by NSF grant CCR-9109785.

†Department of Computer Science, University of Colorado, Boulder, CO 80309-0430 (crivells@cs.colorado.edu and jessup@cs.colorado.edu).

intervals, but Simon [10] proves that bisection is not the optimal method for computing one eigenvalue using a single vector processor.

In [4], we show that on distributed-memory multiprocessors, it is more efficient to compute different eigenvalues on different processors than it is to distribute computation of the Sturm sequence. Further, we show that, unlike in the vector processor case, the relative efficiencies of bisection and multisection depend on a number of variables, such as the size of the matrix and parameters of the multiprocessor used. On an Intel iPSC/2 hypercube multiprocessor, bisection is the best choice only for small sized matrices. In general, multisection with a number of sections equal to the number of processors is the fastest method for computing one eigenvalue. In this paper, we review the results for the Intel Touchstone Delta.

2 A Time Complexity Analysis

In this section, we develop an analytical expression for the time required to compute a single eigenvalue by multisection on a mesh multiprocessor. First, we determine the number of iterations needed to extract an eigenvalue from an interval of width l when the interval is split into $p+1$ subintervals at each iteration. If the final interval width turns out to be $\delta = l\epsilon$, where ϵ is a given threshold, it is necessary to carry out at least k serial *multisection* steps, until

$$l/(p+1)^k \leq \delta.$$

Replacing δ with its value $l\epsilon$ and taking the logarithm gives

$$k_p = \lceil -\log(\epsilon)/\log(p+1) \rceil.$$

Note that when $p = 1$, k_1 gives the number of *bisection* steps. This argument is independent of the base of the logarithm, but throughout this paper, we take $\log(x)$ to mean $\log_2(x)$.

2.1 Computation Cost

At each iteration, p division points are determined and the recurrence in equation (1) is evaluated at each point. If the interval endpoints are λ_{-1} and λ_0, the distance between division points is $\Delta = (\lambda_0 - \lambda_{-1})/(p+1)$, and the division points are $\lambda_i = \lambda_{-1} + i\Delta$, $i = 1, \ldots, p$.

The cost of computing these points is $p\omega_1 + \omega_2$, where ω_1 is the time for a floating point addition or subtraction and ω_2 is the time for a floating point multiplication or division. The Sturm sequence evaluation at each point takes n floating point divisions and $2n$ floating point subtractions. In addition, counting the negative terms in the sequence requires n floating point comparisons. Therefore, the total cost for computing a single eigenvalue by the serial multisection algorithm is

$$T_M = k_p * [p * (n * (2\omega_1 + \omega_2 + \gamma) + \omega_1) + \omega_2],$$

where γ is the time for a floating point comparison.

When the p Sturm sequences are evaluated in parallel —one per processor— at each multisection step, processors must communicate to determine the next search interval. Because communication costs can be quite significant in these parallel algorithms, we examine efficient communication schemes in the following subsection.

2.2 Communication Cost

On existing distributed-memory MIMD multiprocessors, the cost of data communication is high in comparison to the cost of floating point computation. In particular, the time for communicating an m-byte message from one processor to a neighboring one is $\beta + m\,\tau$, where the communication startup latency β is generally large in comparison to the transmission time per byte τ and to the time for a floating point operation.

These cost ratios lead us to consider only communication schemes that minimize the number of message startups. On a hypercube, this minimum number is proportional to the dimension d of

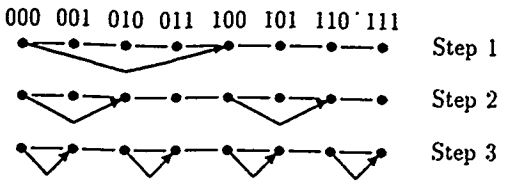

FIG. 1. *Broadcasting a message from node 0 on a linear array (most significant bit to least significant bit). Source: [1].*

the hypercube, with $p = 2^d$ processors [9], and two basic communication mechanisms are possible. The first is based on alternate direction exchange (ADE) as described in [9]. ADE is typically used to accumulate in all $p = 2^d$ processors a vector of length pk whose components are initially distributed evenly among them. In each of d communication steps, the d-cube splits into a different pair of $(d-1)$-cubes. In the first step, corresponding processors in the two cubes exchange their k elements and accumulate a vector of length $2k$. In subsequent steps, the processors exchange and accumulate all previously accumulated data so that in the last step processors send messages of length $2^{d-1}k$ and accumulate the full vector. ADE is the basis of the optimal multisection method on the hypercube [4].

The second approach is a gather-broadcast routine (GB) in which each processor sends its Sturm sequence count to a single master processor. That processor then computes the endpoints of the next search interval and broadcasts them back to all the other processors. The communication cost is

$$2d\beta + (d-1)(4+2*8)\tau,$$

for an integer*4 processor number and real*8 endpoints [4,9].

The total time to compute one eigenvalue using this algorithm is

$$\begin{aligned} T_{GB} &= \{\text{time for one eigenvalue count} + \text{time to find a new interval by gather-broadcast}\} \\ &\quad *\text{number of iterations} \\ &= \{n(2\omega_1 + \omega_2 + \gamma) + 2d\beta + (d-1)20\tau\}\lceil -\log\epsilon/\log(2^d+1)\rceil. \end{aligned}$$

3 On Mesh-Connected Architectures

Although ADE outperforms GB on the hypercube, ADE is intimately tied to the hypercube topology. Because we cannot implement ADE on mesh-connected architectures, we instead use the GB approach for multisection on the mesh. According to [1], there is an optimal broadcast (or gather) algorithm for meshes that does not cause network contention and has the same logarithmic time complexity as for hypercubes. Figure 1 depicts a contention-free broadcast from node 0 to all others on a linear array [1].

If we have a two-dimensional grid of $p = p_1 \times p_2$ points, where $p_i = 2^{d_i}$, $i = 1,2$, it will be necessary to perform $d = d_1 + d_2 = \log(p_1) + \log(p_2) = \log(p)$ steps to complete a broadcast (gather) operation [1]. The basic idea is to partition the two-dimensional array along one dimension, thereby reducing the problem to that for linear arrays. These, in turn, are recursively partitioned, doubling the number of partitions at each step, and creating distinct sub-arrays which can proceed independently with the broadcast (gather) procedure. In this way, a minimum spanning tree broadcast (gather) can be performed on mesh-connected architectures as efficiently as on hypercubes. The only difference is in the way that the minimum spanning tree is derived using the binary representation of the nodes. While the order in which bits are toggled to derive the tree is not important for hypercubes, it is for meshes due to contention problems [1].

Therefore, under reasonable assumptions, the time complexity analysis of the GB multisection approach remains the same on both hypercube and mesh-connected architectures. Figure 2 depicts

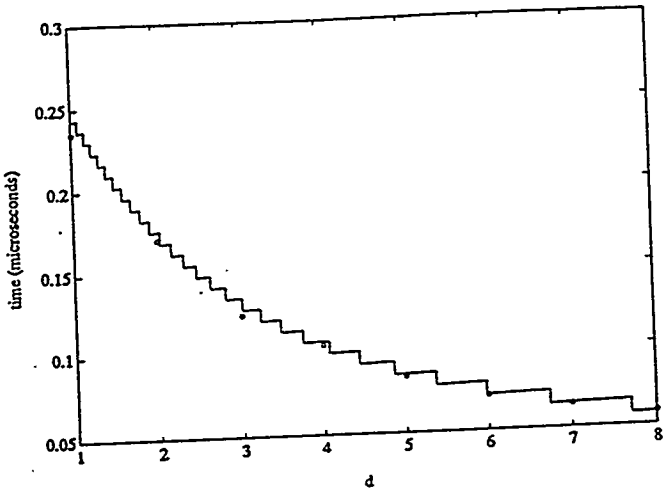

FIG. 2. *Theoretical and actual values of the cost function for computing an eigenvalue of the* $[1,2,1]$ *matrix of order 1000 on the Delta.*

the cost function T_{GB} corresponding to the GB multisection approach for computing the smallest eigenvalue of the 1000×1000 matrix $[1,2,1]$ plotted as a solid line against the continuous variable $d = \log(p)$. On the same plot, it also shows the times measured on the Intel Touchstone Delta (circles). The values of the parameters used in computing T_{GB} are $-\log(\epsilon) = 54$, $\beta = 75$, and $\omega_1 \approx \omega_2 \approx \gamma = 0.1$. These values corresp ond to double precision computation on the Delta. The figure shows good agreement between the theoretical and the actual timings in the range of available processors.

References

[1] M. BARNETT, D. PAYNE, AND R. VAN DE GEIJN, *Optimal broadcasting in mesh-connected architectures.* Tech. Rep. TR-91-38, Dept. of Computer Science, University of Texas at Austin, 1991.

[2] W. BARTH, R. MARTIN, AND J. WILKINSON, *Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection*, in Handbook for Automatic Computation: Linear Algebra, Springer Verlag, 1971, pp. 249–256.

[3] H. BERNSTEIN AND M. GOLDSTEIN, *Optimizing Givens' algorithm for multiprocessors*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 601–602.

[4] S. CIAVELLI AND E. JESSUP, *Optimal eigenvalue computation on distributed-memory mimd multiprocessors*, Tech. Rep. CU-CS-617-92, Dept. of Computer Science, University of Colorado, 1992.

[5] W. GIVENS, *Numerical computation of the characteristic values of a real symmetric matrix*, Tech. Rep. ORNL-1574, Oak Ridge National Laboratory, 1954.

[6] H. HUANG, *A parallel algorithm for symmetric tridiagonal eigenvalue problems*, CAC Document No. 109, Center for Advanced Computation, University of Illinois, 1974.

[7] I. IPSEN AND E. JESSUP, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, SIAM J. Sci. Stat. Comput., Vol. 11, No. 2, (1990), pp. 203–229.

[8] S. LO, B. PHILLIPE, AND A. SAMEH, *A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. s155–s165.

[9] Y. SAAD AND M. SCHULTZ, *Data communication in hypercubes*, Research Report 428, Dept Computer Science, Yale University, 1985.

[10] H. SIMON, *Bisection is not optimal on vector processors*, SIAM J. Sci. Stat. Comput., 10 (1989). pp. 205–209.

[11] J. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.