MATRIX DIAGONALISATION BY TRANSPUTERS

Terry Bossomaier & Richard Brent
Computer Sciences Laboratory
Australian National University

We discuss the efficiency of transputer networks for diagonalising real symmetric matrices. Brief consideration of the various methods available suggests that only the Jacobi method adapts easily to a concurrent sequential process architecture. Suitable organisations of transputer networks are evaluated in terms of the ratio of communication overhead to computation. The maximum efficiency that can be achieved by the theoretical best possible transputer implementation of the parallel Jacobi algorithms of Brent & Luk (1985) is calculated.

INTRODUCTION

At present the most efficient methods for diagonalising dense symmetric matrices appear to be those based on the Householder/QR algorithms (Golub & van Loan, 1983) but these methods do not adapt easily to a parallel architecture. The Jacobi method, however, adapts rather easily (Brent & Luk, 1985). Nevertheless, its application in concurrent sequential process architectures is not necessarily very effective, particularly when each processor is very powerful relative to its communication speed: such is the case with the *INMOS* T800 transputer. In fact if we have a lot of matrices to diagonalise the most efficient strategy is to use a serial algorithm, such as the Householder combination, in each transputer and give each such processor a whole matrix!

The performance for arbitrarily large matrices can be estimated theoretically. In an ideal architecture, communication would occur concurrently with computation and the efficiency would be unity, but in practical cases, this is often not the case. The transputer, in fact, uses the same CPU to control communication as for its arithmetic work: thus the sending or receiving of a message suspends all the other processes. Thus we can define a theoretical efficiency, η, as

$$\eta = \frac{t_{calc}}{(t_{comm} + t_{calc})} = 1 - \frac{t_{comm}}{(t_{comm} + t_{calc})}$$

where t_{comm} and t_{calc} are the times spent communicating and calculating respectively. We refer to the quantity $\frac{t_{comm}}{(t_{comm}+t_{calc})}$ as the *communication defect.* Determination of the efficiency is important, because an operation such as matrix diagonalisation is rarely done just by itself, but in the context of a broader enterprise, perhaps involving more than one matrix. It may often be desirabable to let a single transputer do the all the matrix operations, or to give several transputers each a separate matrix to work on.

Summary of the Brent & Luk (BL) Algorithm

Brent and Luk (1985) have described a modification of the classical Jacobi algorithm which is suitable for implementation on a systolic array or a network of transputers. When finding the eigenvalues of an N by N matrix, the Brent-Luk algorithm uses $(N/2)^2$ processes or "virtual processors" $P_{i,j}$, $i,j = 1,\ldots,N/2$ which can be imagined to be arranged in an $N/2$ by $N/2$ square array. (For the sake of brevity we assume N to be even and do not discuss the computation of eigenvectors; the extension to odd N and/or the computation of eigenvectors is straightforward.) It is important to note that each process $P_{i,j}$ needs to communicate only with its four nearest neighbours $P_{i\pm1,j\pm1}$.

Each process $P_{i,j}$ performs some simple computations on a 2 by 2 matrix and then exchanges information with its neighbours. After $N-1$ such steps one "sweep" of Jacobi's algorithm has been performed; typically about 6 sweeps are sufficient for convergence.

Details of the floating-point computations performed by each process $P_{i,j}$ are described in Brent and Luk (1985). For our purposes it is sufficient to note that in each step the *diagonal* processes (i.e. those $P_{i,j}$ with $i = j$) perform 2 square roots, 3 divisions, 4 multiplications and 6 additions [1] while the off-diagonal processes (i.e. those $P_{i,j}$ with $i \neq j$) perform 16 multiplications and 8 additions. After the computations for a step are performed, each process $P_{i,j}$ sends up to 8 real numbers to its horizontal neighbours $P_{i,j\pm1}$ and receives up to 6 real numbers from its horizontal neighbours, then communicates similarly with its vertical neighbours $P_{i\pm1,j}$. [2] Thus the number of real numbers communicated by each process is comparable to the number of floating-point operations that it performs.

If $(N/2)^2$ transputers are available then each process $P_{i,j}$ can be placed on its own transputer, with hardware links to the transputers running the neighbouring processes $P_{i\pm1,j\pm1}$. However, it is also possible to place several processes on each transputer - the block BL algorithm.

Theoretical Efficiency

The present discussion assumes a square symmetric, $N \times N$, matrix. The eigenvectors are not considered here, but their determination makes it difficult to make much use of the symmetry. Thus we do not take account of symmetry in the subsequent analysis. The discussion is simplified by assuming that N is an even power of 2. Other cases may show some small losses in efficiency relative to this case. Since the sweeps have to proceed sequentially, we can ignore multiple sweeps entirely and concentrate on the relative efficiencies of

---

[1] Variations exist; for example, it is possible to avoid square roots entirely.

[2] The diagonal and boundary processes differ slightly from the other processes.

a single sweep. Results are given in a general form and specifically for the T800 transputer. Although the arguments are not specific to the transputer, for clarity we refer to the virtual 2×2 processors of the BL algorithm as the unit processors and the actual processors as transputers.

Block form of the BL algorithm

Let the number of processors be P^2, with each processor holding a sub-matrix of size $m \times m$, giving

$$N = mP$$

The diagonal processors have to compute $\frac{m}{2}$ rotation parameters and carry out $(\frac{m}{2})(\frac{m}{2}-1)$ rotations of off-diagonal 2 blocks. Since computing the rotation parameters is more time consuming than actually doing the rotation, the diagonal processors are rate determining. Letting t_{rp} be the the time to calculate the rotation parameters and t_r the time to do a rotation, then the diagonal block processor takes time T_d given by

$$T_d = (\frac{m}{2})(t_{rp} + (\frac{m}{2} - 1)t_r)$$

The inner diagonal processor have to broadcast $\frac{m}{2}$ copies of $\cos\theta$ and $\sin\theta$, where θ is the Givens rotation angle. Let t_{cr} be the time to send or receive a single real number. The communication overhead is then $4mt_{cr}$. We then have to permute the rows and columns. For a central processor this involves moving m real numbers across each of four interfaces. Thus we arrive at the following results

$$t_{comm} = 12mt_{cr}$$

$$t_{calc} = T_d$$

giving the communication defect ϵ

$$\epsilon = \frac{16t_{cr}}{(t_{rp} + (\frac{m}{2} - 1)t_r)}$$

To determine performance figures for the T800 we need estimates of t_{comm}, t_r and t_{rp}. The link speed of 20Mbits/sec. gives a communication time of approximately 32 processor cycles for each word transmitted. Hence (in processor cycles)

$$t_{cr} = 32w$$

with w the number of words transmitted i.e.

$$t_{comm} \simeq 384m$$

Counting up additions, multiplications and square roots gives us

$$t_{rp} \simeq 400$$

$$t_r \simeq 240$$

Both the above are under estimates: the communication does not include control bits and the calculation ignores addressing overheads.

$$t_{calc} = (\frac{m}{2})(400 + 240(\frac{m}{2} - 1))$$

giving

$$\epsilon \simeq \frac{1}{(1+m)}$$

Obviously for small m the communication penalty is draconian but, for large matrices, such as one might wish to use a multi-processor network for, we can get very good results - for a 400 square matrix on 4 transputers, we would have $\epsilon \simeq 0.01$ i.e. efficiency of about 99%. Note that using $O(N^2)$ processors, as in the basic BL algorithm gives a very low efficiency of about 50%.

DISCUSSION

The algorithms considered here represent a reasonably effective use of transputer networks for finding matrix eigenvalues. The efficiency is rather poor for small matrices, but for large matrices becomes acceptable. One difficulty considered is the uneven computational load on transputers holding diagonal versus those holding off-diagonal elements. But the inefficiency inherent in keeping the off-diagonal processors idle while the rotational parameters are calculated, can potentially be overcome: where a transputer has only off-diagonal 2×2 processors, it should simply have more of them. The precise trade-off remains to be investigated.

REFERENCES

Brent R.P. & Luk F.T. (1985) "The solution of singular value and symmetric eigenvalue problems on multi-processor arrays", *Siam. J. Sci. Stat. Comput.* **6**(1), 69-84.

Golub G.H. & van Loan C.F. (1983) "Matrix computations", JohnsHopkins University Press,Baltimore, U.S.A.