A Novel Parallel Direct LU Decomposition Matrix Solver for Banded Matrices

1 INTRODUCTION

The need to invert matrices corresponding to large systems of simultaneous equations is commonplace in the field of computer simulation of physical systems. Such matrices can arise from the discretisation of multi-dimensional continuous problems posed in the form of partial differential equations.

At present computing power imposes severe restrictions on the scale of simulations, though it may be argued that this will always be the case as problem size can in practice expand far beyond any arbitrary limit.

To reduce the time taken by any particular simulation three approaches seem possible:-

1. buy a faster scalar computer.

2. buy a computer and software that can exploit any parallelism present in the problem and modify the code so that parallelism can be achieved.

3. build a dedicated systolic array.

Of the three options the first is by far the easiest if budgetary considerations are ignored and technology allows, however, given an identical technology a computer that can exploit parallelism will always be significantly faster provided suitable parallel algorithms exist for the problem in hand. Or alternatively parallel computing may reduce the cost for a given computational power.

Recently parallel computing has become a realistic i.e. relatively inexpensive option with the advent of the INMOS Transputer[1] and provision for mounting transputer hardware within a host scalar machine eg PC, Sun, Apollo etc. Along with the hardware suitable software in the form of language compilers is also begining to appear in particular "3L Parallel C" [2]

With regard to the systolic array approach[3] The main drawback is that such arrays are hardwired and do not offer the flexibility of a programmable processor such as the transputer.

In this technical note a parallel algorithm based on LU decomposition for the direct solution of a system of linear banded simultaneous equations will be described, iterative solution methods will not be discussed apart from the following comment.

Iterative methods can be relatively easily parallelised with very nearly linear speedup in the time taken to perform an iteration, however, as convergence cannot be guaranteed situations can, and therefore unfortunately do, arise in which a very slow convergence rate couples with an arbitrarily fast iteration time to give the overall result that the computations get nowhere very fast. In contrast direct methods always give an answer in the same time for a given problem size regardless of the nature of the problem and therefore an efficient parallel direct solver is a desirable tool.

2 METHOD

LU decomposition serves to invert the matrix equation

$$Mx = y \qquad [1]$$

where:

M is the banded matrix.

x is the unknown vector.

y is the known vector.

in the following way, let $M = LU$ where:

L is a lower triangular matrix

U is an upper triangular matrix with $u_{ii} = 1.0$

both L and U have the same bandwidth as the original matrix M. We can define a vector v such that

$$v = Ux \qquad [2]$$

equations 1 and 2 imply that

$$Lv = y \qquad [3]$$

Equation 3 may easily be inverted due to the lower triangular property of L. Thus vector v may be determined and once this is achieved then 2 may be used to obtain x as U can easily be inverted due to its upper triangular property.

In this way the inversion of M is seen to reduce to the problem of finding its LU factors which, once found, need to be stored for subsequent solution of 1 via the process just described.

Existing methods of parallel LU decomposition, for banded matrices, seem to make inefficient use of the available processors[4] as they dissect the matrix into square areas and assign individual processors to each area, see figure 1. Solution

starts in the top left corner where processor A has all the information it needs to factor its assigned area of the matrix. Processor A has then to pass its L factor to processor B and its U factor to processor C before both B and C can simultaneously begin work on their allotted tasks. Once A has communicated with B and C its task is complete and it remains idle. Thus the process continues with L factors propagating from left to right between processors and U factors propagating down the array of processors. At any given time the active processors lie in a diagonal line perpendicular to the main diagonal of the matrix. This technique is called the multi- frontal technique as at any given time the boundary between the known and unknown elements of L and U forms a stepped wavefront along the boundary line of active processors.

An immediately obvious way of significantly improving the efficiency of the above algorithm is to scan a line of processors down the matrix and arrange for each processor to store the L and U elements in its path. However there remains an inherent inefficiency that has not yet been highlighted. When computing LU factors the processor assigned to the region of the main diagonal of the matrix has much more work to do as it must perform vector dot products on vectors with lengths of the order of the band width whilst the processors at the band edges have only dot products with lengths of the order of the band width divided by the number of processors.

This unequal load-sharing implies that the central processor constitutes a bottleneck in the solution process as all other processors must wait for it to complete its current task before they can proceed.

To overcome this load-sharing problem it is proposed in this technical note that individual processors should be assigned to segments of a single diagonal wavefront, that scans down the main diagonal of the matrix as indicated in figure 2. The segment lengths are adjusted so that the computational load is shared as evenly as possible between the available processors. Individual L and U vectors would enter at opposite segment edges progress along the segment as the wavefront advances and exit at the edge opposite to their entry.

It should be noted that the proposed algorithm suffers from a reduced compute-to-communicate ratio when compared to the multi- frontal technique as communication has to take place each time the wavefront moves one node down the matrix diagonal. Whereas the multi-frontal communicates L and U vectors in blocks corresponding to the number of rows or columns covered by each processor.

It takes time to perform a communication between processors, time which is thus denied to the process of actually performing the numerical calculation. However if the number of processors is increased for a given problem, the multi-frontal compute-to- communicate ratio will tend towards that of the proposed algorithm so that the much improved load-sharing of the new algorithm will at some point

manifest itself. Indeed it may be the case that the improved load-sharing more than compensates for the extra communication overhead although this has not been investigated as the author does not have access to a multi- frontal code.

It is possible to combine even load-sharing with a high compute-to- communicate ratio in a modified version of the proposed algorithm. This modified algorithm will be discussed in a later section.

3 PERFORMANCE

The algorithm described above has been implemented using the '3L Parallel C' compiler running on 'INMOS T800 Transputer' microprocessors clocked at 20 Mhz. The Transputer is especially suited to parallel processing due to is four on-chip fast links which enable efficient (20Mbit/sec) nearest neighbour communication between processors and its on chip floating point co-processor which is capable of 1.5 Mflop operation.

In order to make a fair comparison of any speedup obtained from the parallelisation process, results must be compared to the best available sequential algorithm. It is uncertain which sequential algorithm is best so this technical note makes use of the sequential form of the proposed parallel algorithm. As this may not be the best algorithm the absolute time for sequential solution of a given problem will be quoted in addition to the relative speedup when presenting results.

When reading the literature it should be noted that any claimed speedup that is super-linear with processor number implies either that the chosen sequential algorithm is not the best known or that the claimant has invented a better sequential algorithm in the course of his work on the parallel algorithm.

3.1 Sequential Algorithm Results

A sequential form of the proposed algorithm has been implemented and figure 3 shows the time taken to solve trial problems, on a single transputer, as a function of the number of variables. For variable numbers greater than 2300 the data was extrapolated as these problem sizes require more than the available 2Mbyte memory associated with each of our transputers. The extrapolation was performed by putting a line of slope four through the 1600 variable point.

From this data it can be determined that a performance rating of 0.45 Mflops was achieved from a single Transputer.

The same sequential code has been run on a number of other computers in order to assess the transputers performance and the results are dislayed in Table 1 for

both a 1600 (bandwidth = 40) and, where available memory was adequate, a 4900 (bandwidth = 70) variable problem. Where memory was not adequate a projected solution time is given, indicated by a question mark (?), based on a second power ratio of work to variable number.

Computer	1600 variable	4900 variable
Transputer	14.5	136.0?
Apollo DN4000 + fpa1	18.5	162.5
Apollo DN10000	3.2	30.2
Sun 4	8.0	64.5
IBM 4381	8.98	69.98
Vax 11/780	43.5	408.0?

Table 1. Sequential algorithm solution times for various computers

3.2 Parallel Algorithm

The parallel algorithm requires implementation on a linear array of processors with I/O achieved via a connection to the central processor, figure 2. An odd or even number of processors could be used, however the central processor would require a slightly different form of code for both cases. In this work it was decided to use an odd number of processors only.

Full implementation of the parallel algorithm requires a set of five sub-programs, the main program resident on the central processor, special sub-programs for both L and U band edge segments resident on the outer processors and two sub-programs for interior L and U segments resident on all L and U interior processors respectively as depicted in figure 2.

3.2.1 Determination of Optimum Segmentation

In this section we shall investigate the problem of determining the optimum segmentation of the band for equal load-sharing between the available processors.

As stated earlier, the work required to compute an L or U element is not constant across the band. The computation consists of vector dot products where the length of the vectors is equal to the bandwidth (W), on the main diagonal, and zero at the band edges. Communications also follow this pattern. Thus the work can

be represented as the area of an isosceles triangle of base 2(W + 1) and height (W + 1) as in figure 4.

The segmentation problem thus reduces to that of dividing this triangle into n, equal area, vertical slices where n is the number of processors. This results in the following formula for the optimum segmentation:-

$$S_i = \left(1 - \sqrt{1 - \frac{(2i-1)}{n}}\right)(W+1) - 1/2 \qquad [4]$$

where S_i is the i th segment (S_1 is the central segment)

Note that S_i will in general be non-integer , values of S_i were rounded down to the nearest even integer in this work. The restriction to even integer segment widths arises from the diagonal nature of the wavefront, a single step along the wavefront corresponds to crossing two matrix diagonals see figure 2. If the bandwidth is odd the outer segments only are odd.

Thus the central, first, processor covers $2S_1 + 1$ diagonals as it spans the main diagonal. Whilst the i th processor in both halves of the band cover $S_i - S_{i-1}$ diagonals each.

3.2.2 Results

Figure 5 plots the speedup achieved relative to the sequential algorithm (left hand axis) and Mflop rating (right hand axis) as a function of the number of Transputers and the problem size. The maximum problem size was limited to approximately 6000 variables by the available memory associated with each transputer i.e. 2 Mbytes. The bandwidth of all test problems was equal to the square root of the number of variables.

Included in the figure is a projection, based on the trends exhibited by the results for smaller numbers of variables, to 14400 variables. The projection suggests that a speedup of 13 i.e. 6.0 Mflops, could be obtained from 24 transputers. In making this projection an attempt was made to avoid being over optimistic.

The efficiency of the algorithm can be gauged by reference to the 100% and 50% efficiency lines drawn in figure 5 and varies from better that 60% (when the number of processors is small relative to the bandwidth) to 40% when the optimal central segment reduces in width to its practical minimum value of 2.

From the above discussion of efficiency its is clear that the number of processors that can be effectively assigned to matrix inversion is limited by the band width.

the exact number can be determined from equation 4 by setting $S_1 = 2$ and calculating n via:-

$$n = \frac{1}{1 - \left\{ \frac{W - 3/2}{W + 1} \right\}^2} \qquad [5]$$

Table 2 displays data from a problem size of 4900 variables, illustrating the effect of varying the segmentation for different numbers of Transputers. Its is clear that the calculated optimum is the practical optimum and that a considerable time advantage is gained over the equivalent multi-frontal segmentation where each processor computes over equal segment lengths, though of course the compute to communicate ratio is worse in this case compared to a proper implementation of the multi-frontal algorithm.

Transputers	Segmentation	Optimal ?	Time sec
5	26, 8	no	48.19
5	24, 6	yes	43.47
5	22, 6	no	45.43
7	32, 18, 6	no	41.19
7	30, 16, 4	no	35.87
7	28, 14, 2	no	37.15
7	28, 14, 4	no	37.10
7	30, 14, 4	no	35.55
7	32, 16, 4	yes	35.47
7	50, 30, 10	no, multi-frontal	51.01

Table 2. Results for a 4900 variable, bandwidth = 70 problem.

The time taken for sequential solution of this 4900 variable problem is projected to be approximately 136 seconds.

These results should be compared to those displayed in table 1, such a comparison shows that 7 transputers come close to equalling the speed of an Apollo DN10000. For larger problems the 7 Transputer array would be the fastest of the computers tested.