Applied Mathematics & Information Sciences **2**(2) (2008), 225–235
– An International Journal

Parallel Solution of Linear System of Equations on Transputer Array

S. H. Abbas

Department of Mathematics, College of Science, University of Bahrain

Sakhier, P. O. Box 32038, Bahrain

Email Address: dr_salman121@hotmail.com

Received November 27, 2007; Revised January 30, 2008; Accepted February 22, 2008

This paper describes the implementation in Occam of linear system of equations solver on a network of transputer. Using Gaussian elimination with partial pivoting, we present a general solution. The experimental and the theoretical timings are given. Analysis of the optimum number calculations of processors also is presented.

Keywords: Gaussian elimination, experimental and theoretical timings, optimum number of processors.

2000 Mathematics Subject Classification: 65F10, 65F50.

1 Introduction

The numerical methods for parallel solution of linear system of equations are well established techniques in literature. In the last two decades many papers have appeared on this topic, see, for example [1]–[22] and references therein.

Let us consider the linear system of equations

$$A\underline{x} = \underline{b}, \tag{1.1}$$

where A is a real square dense matrix of size $n \times n$, $\underline{x}$ and $\underline{b}$ are vectors of size $n \times 1$. The reader is assumed to be familiar with the Gaussian Elimination algorithm. Details of the sequential algorithm can be found in Abbas [1]. The elimination procedure has been split into two distinct sections:

(i) Reduction of the matrix to upper triangular form.

(ii) Forward elimination and backward substitution of the right hand side vector.

The cost of the sequential matrix reduction algorithm is of order n^3 and it offers good scope for parallelism.

A sequential implementation of the second part has a cost of order n^2. For applications where this is performed once each time the matrix reduction is performed, the additional

cost involved will be small for large n, that is in situations where it will be performed many times for each matrix reduction, e.g. for iterative inverse power methods. The total cost of the repeated evaluation of solution vectors will become significant compared with the initial cost of matrix reduction. Hence the second part should also perform the forward substitution and backward elimination of the vector in parallel to keep its overall cost down.

A routine to perform Gaussian Elimination based upon a distribution of columns to transputers is already available [1]. For comparison purposes this implementation distributes rows to transputers.

2 Network Topology

The routine executes on a ring of p slave processors. This simple topology was chosen in preference to a mesh topology because it simplifies the communications required. A ring instead of a chain is used so that communication during the algorithm can all be in one direction and thus communication conflicts are avoided. The slaves are numbered from 0 on the right up to $(p-l)$ on the left. The leftmost slave is linked to a master processor and has a return link to slave 0.

3 Algorithm Description

When the routine is called the master processor first sends out the rows of the matrix such that slave i receives rows i, $i+p, i+2p$, etc. This method of distribution helps to keep an even workload between the slaves throughout execution. To make this operation as efficient as possible whilst one row is being output to the right by a slave, it will also be receiving the next row from the left. In this way the total time for a row to be output and the next row to be input is only slightly greater than that required simply to output the row. All the rows for slave 0 are sent first, so that whilst rows are being sent to higher numbered slaves, those with lower numbers can begin their calculations.

Each slave executes the same main program $-a$ loop for each column, k, of the matrix, within which the pivot element for that column is chosen and pivoting performed.

For each loop iteration, the slave which will be first to start is assumed to be that to the left of the slave which on the last iteration owned the pivot row. In general, assuming that a row swap was required and that there is an even distribution of the workload, this will be valid. However, if a row swap was not made in the previous iteration then the slave, which owns the last pivot row, should be the first to start the next iteration.

Within each loop, the best pivot element must be chosen by communication between the slaves, and then the rows of the matrix held by each slave updated using the chosen pivot row.

Choosing the best pivot element is divided into two parts. Firstly, a slave will choose its element with largest magnitude in column k as a pivot. This is compared with the best suggestion of previous slaves, input from the right, if there are such slaves, and the best of these is output left as a new suggested pivot. The last slave hence outputs the best pivot of column k which is then passed left round the ring to all slaves. The pivot row numbers chosen at each step are recorded by each slave for subsequent use in the formation of the result vector.

Secondly, the owner of the best pivot row outputs the slice of that row from element k upwards. Each slave inputs the row and passes it on if necessary. Also if a row swap is required because the best pivot row was not the k^{th} row, then the k^{th} row owner will output the k^{th} row to the left before inputting the pivot row. By physically exchanging rows between slaves the overall workload balance is maintained.

The pivoting then proceeds with each slave updating those rows it holds from element $k + l$ onwards, and storing the calculated factors in the k^{th} column for later use:

Calculate factor: $a_{ik} = a_{ik}/a_{kk}$ for all $i > k$,

Perform pivoting: $a_{ij} = a_{ij} - (a_{ik} \times a_{kj})$ for all $i > k, j > k$.

When this is complete each slave in the network has a subset of rows of the reduced matrix and a vector of the pivot rows chosen at each pivoting step.

3.1 Calculation of Result Vector

At the start of the second part the right hand side (rhs) vector, $\underline{b}$, is passed out to the network. As for the first routine, the initial distribution of vector elements is such that slave i receives elements i, $i + p, i + 2p$, etc. This distribution ensures that all data required to perform pivoting on a slave's rhs elements (with the exception of the pivot element for each step) is already available on that slave.

The calculation of the solution vector for the simultaneous linear equations is split into two sections:

(i) Pivoting as for the matrix reduction, followed by

(ii) Backward substitution to yield the result vector $\underline{x}$.

The first section is performed in a manner very similar to that employed for the matrix reduction. For each column k of the matrix, the k^{th} row owner broadcasts the k^{th} element of $\underline{b}$ to all the slaves. If the pivot row chosen in the matrix reduction was not also the k^{th} row then the corresponding elements of the right hand side vector, $\underline{b}$, are exchanged. This may require communication between slaves to accomplish. The slaves then perform pivoting on their rhs elements, utilizing the factors, a_{ik}, calculated in the matrix reduction routine

$$b_i := b_i - (a_{ik} \times b_k) \qquad \text{for all} \qquad i > k.$$

multiply two REAL32 elements of a vector one million times. The other arithmetic operations $(+, -, /)$ give slightly different timings, so a better measure for arithmetic cost might be the cost of the entire central arithmetic operation, e.g.

$$a[i][j] := a[i][j] - (a[i][k] \times a([k][j]).$$

However, this measure would not be generally applicable to theoretical cost expressions for other algorithms, which did not include this arithmetic operation.

The time taken to pass a vector of 1000 elements down a hardware link 1000 times was measured in deriving a value for t_c.

It was found that the values of t_f and t_c depend not only upon the type of transputer in use $(T4, T8)$ and the link speed settings, but also varied between similar systems of $T4$ transputers. For consistency, all parallel algorithm timings were therefore made on the RSRE Protonode, for which these values were measured as

$$t_f = 17.6 \text{ microsecs} \quad \text{and} \quad t_c = 8.4 \text{ microsecs} \quad \text{and hence the ratio,} t_f/t_c = 2.1.$$

The properties of the parallel algorithm that we are interested in are the speedup Sp and the efficiency eff. The speedup of a parallel algorithm is defined as, cost of sequential algorithm / cost of parallel algorithm.

From this the efficiency is derived

$$e_{ff} = S_p/p.$$

4 Matrix Reduction

4.1 Cost of parallel algorithm

For a matrix of size n the cost of performing the arithmetic operations in the central pivoting loop is given by

$$\frac{t_f}{6}\left(4n^3 - 3n^2 - n\right). \tag{4.1}$$

Assuming that the arithmetic is well balanced between processors the total time taken by processor to update its rows is

$$\frac{t_f}{6p}\left(4n^3 - 3n^2 - n\right). \tag{4.2}$$

The time taken for the last slave to receive its rows during initialization is: $n^2 t_c$. since each communication involves the transfer of n items. The time spent during the algorithm to evaluate pivot elements, distribute pivot rows and perform row swaps is

$$t_c\left[2p(n-1) + (p-1)(\frac{n^2+n}{2})\right], \tag{4.3}$$

of which $2p(n-1)t_c$ is the total cost of choosing the best pivot elements, and $(p-1)[(n^2+n)/2]t_c$ is the total cost of broadcasting the pivot rows. Total communication time is hence given by

$$\frac{t_c}{2}\left[5pn + n^2p - n - 4p\right]. \tag{4.4}$$

So, the total algorithm cost for a matrix of size n and p slaves is

$$C_{red} = \frac{t_f}{6p}\left(4n^3 - 3n^2 - n\right) + \frac{t_c}{2}\left(5pn + n^2p - n - 4p\right). \tag{4.5}$$

4.2 Optimum number of processors

The optimum number of slaves P_{red} to give the minimum algorithmic cost C_{min} is found when

$$dC_{red}/dp = 0, \tag{4.6}$$

i.e.

$$\frac{t_f}{6P_{red}^2}(4n^3 - 3n^2/2 - n) = \frac{t_c}{2}(n^2 + 5n - 4), \tag{4.7}$$

$$P_{red}^2 = \frac{t_f}{3t_c}\left(\frac{4n^3 - 3n^2 - n}{n^2 + 5n - 4}\right). \tag{4.8}$$

So for large $n(n >> 1)$ we get

$$P_{red} = \frac{1}{3}\sqrt{4n(tf/tc)}. \tag{4.9}$$

5 Calculation of Result Vector

5.1 Cost of sequential algorithm

The number of arithmetic operations for the forwards elimination is $n(n-l)$. The number of arithmetic operations in the back substitution is n^2. Hence the total cost to calculate result vector is n_2 given by $n(2n-1)t_f$.

5.2 Cost of parallel algorithm

Cost of the initial distribution of the right hand side vector n $(2p-l)t_c/p$. The cost for the forward elimination of the vector has two components:

1. Cost of passing round the pivot elements and swapping: $(n-1)ptc$.
2. Total cost in performing the pivoting $n(n-l)tf/p$.

The cost of the back substitution is also of two parts:

1. Initial calculation of last element: $t_f + t_c$.

2. Cost of loop to evaluate remaining elements

$$(n-l)nt_f/p + (n-1)(2t_f + t_c). \tag{5.1}$$

Returning the result vector to the master costs: $n(2p-l)t_c/p$. Hence the total cost for the parallel result vector calculation is

$$C_{rhs} = 2n(nt_f - t_f - t_c)/p + n(5t_c + 2t_f) - t_f + pt_c(n-l). \tag{5.2}$$

5.3 Optimum number of transputers

The optimum number of slaves P_{rhs} to give the minimum algorithmic cost C_{min} is found when

$$dC_{rhs}/dp = 0, \tag{5.3}$$

i. e.

$$-2n(nt_f - t_f - t_c)/P_{rhs}^2 = t_c(n-l).$$

So,

$$P_{rhs} = \sqrt{2ntf/tc - 2n/(n-l)}. \tag{5.4}$$

For large $n(n >> 1), n > t_f/t_c$ we get

$$P_{rhs} = \sqrt{\frac{2nt_f}{t_c}}. \tag{5.5}$$

6 Experimental Timings

Tests were made on networks between 2 and 16 slaves for matrix sizes from 16 to 256. The test matrices used were initialized by filling with pseudo-random REAL32 values. Two separate times were recorded: one for the reduction of the matrix to upper triangular form, and also other for calculating the result vector.

A sequential implementation of the algorithm was written and the costs of the matrix reduction and result vector calculation measured for various matrix sizes. Using these results both the experimental and theoretical speedup and efficiency of the two parts of the parallel algorithm were calculated (see Tables 7.1, 7.2 and 7.3).

7 Comparisons with Other Implementations

As an indication of the performance of the algorithm and of the suitability of transputer networks for parallel computation, the results are compared with other implementations on various systems. The time used for the algorithm is the combined time for the matrix reduction and the calculation of the result vector. A similar algorithm has been implemented in [12].

George on an Intel Hypercube [12], was based upon distribution of rows and uses a spanning tree structure for communications between processors. The speedups and efficiencies for large matrices using small numbers of processors showed reasonable agreement but for problems where communication time is significant, i.e. for larger networks and/or smaller matrices, the transputer- based algorithm gave better results. The timings were 2.5 to 3 times slower than the transputer results, in agreement with the performance figures for the Hypercube which are about 2.5 times slower than those for a T4 transputer.

Table 7.1: Hypercube and Transputer timings

	n	p	time(Secs)	speedup	efficiency
Hypercube	200	16	28.4	7.2	0.45
Transputer	2000	16	10.6	10.0	0.63

An implementation in Ada running on a Sequent Balance has been written by Abbas [1]. Across the range of problem sizes, this was tested with the transputer implementation and had efficiency values about 10% better. Note also that the timings are all about 10 times slower than for the transputer implementation due to the lower performance of the Sequent Balance

Table 7.2: Sequent Balance and Transputer timings

	n	p	time(Secs)	speedup	efficiency
Sequent	100	6*	36.1	4.1	0.69
Transputer	100	6	2.9	4.7	0.79

Sequent result was obtained from a configuration of 6 processors, running 13 separate Ada tasks.

Table 7.3: Transputer timings

	n	p	time(Secs)	speedup	efficiency
Abbas	256	16	25.1	8.9	0.55
Transputer	256	16	20.2	11	0.69

A comparison with the implementation by Abbas [1] on a transputer network using a column distribution to processors proved interesting. It was expected that results would be roughly comparable as both implementations use similar topologies. However, final results show that across the range of matrix sizes and numbers of transputers tested this implementation was consistently better than that due to Abbas [1].

Table 7.4: Matrix Reduction timings

Size of the matrix	Matrix Reduction	Calculation of result vector
16	0.06	0.01
32	0.45	0.04
64	3.51	0.17
128	27.69	0.69
256	220.03	2.78

Table 7.5: Theoretical and Experimental Efficiencies I

Size of matrix	N. Processors	Theor. efficiency	time (secs)	Exp. speedup	Efficiency
16	2	0.05	0.57	0.57	0.89
	4	0.04	0.36	0.36	0.67
	8	0.04	0.18	0.18	0.34
	16	0.05	0.07	0.07	0.11
32	2	0.27	0.83	0.83	0.95
	4	0.18	0.62	0.62	0.83
	8	0.15	0.37	0.37	0.54
	16	0.17	0.17	0.17	0.23
64	2	1.91	0.92	0.92	0.98
	4	1.09	0.81	0.81	0.91
	8	0.75	0.59	0.59	0.72
	16	0.70	0.31	0.31	0.39
128	2	14.28	0.97	0.97	0.99
	4	7.53	0.92	0.92	0.96
	8	4.50	0.77	0.77	0.84
	16	3.42	0.51	0.51	0.57
256	2	110.53	1.00	1.00	0.99
	4	56.12	0.98	0.98	0.98
	8	30.84	0.89	0.89	
	16	19.86	0.69	0.69	0.73

8 Conclusions

The theoretical model for the matrix reduction predicts the efficiency well for larger matrices. For smaller matrices the model is not so good, due to some extent to the uneven distribution of working rows for much of the algorithm.

The efficiency of the calculation of the result vector is much less than that predicted suggesting that the model does not take account of some algorithmic costs. A contributing

Table 7.6: Theoretical and Experimental Efficiencies II

Size of matrix	N. Processors	Theor. efficiency	time (secs)	Exp. speedup	Efficiency
16	2	0.009	1.18	0.59	0.79
	4	0.007	1.51	0.38	0.58
	8	0.007	1.51	0.19	0.34
	16	0.009	1.18	0.07	0.15
32	2	0.032	1.35	0.67	0.88
	4	0.021	2.06	0.67	0.88
	8	0.017	2.54	0.32	0.50
	16	0.018	2.40	0.15	0.25
64	2	0.118	1.47	0.74	0.94
	4	0.067	2.9	0.65	0.85
	8	0.46	3.77	0.47	0.67
	16	0.041	4.23	0.26	0.40
128	2	0.455	1.53	0.76	0.97
	4	0.239	2.91	0.72	0.92
	8	0.142	4.89	0.61	0.80
	16	0.105	6.62	0.41	0.57
256	2	1.786	1.56	0.78	0.98
	4	0.903	3.08	0.77	0.96
	8	0.492	5.65	0.71	0.86
	16	0.316	8.79	0.55	0.73

factor is the domination of communication in this section of the algorithm, which is difficult to model accurately, see Table 7.4.

It was difficult to account in a model for the idle time while processes synchronies for link communication. So instead of trying to find an expression for the total processing time for an individual transputer, which will involve estimating idle time, the model developed predicts the global cost of performing sub-sections of the algorithm. For example the time taken for the initial distribution of rows to the slaves in the network. But this method introduces the problem of deciding upon the most suitable start and end point of these sub-sections: for the initial row distribution, is the end marked when the last transputer receives its final row and hence when all the slaves have completed that section of code? Or when the first slave has received all its rows and begins finding the pivot estimate whilst the other slaves still await their rows, or indeed some other mid-way point?

In spite of these difficulties, the dominant matrix reduction is predicted well, especially for larger matrices allowing predictions to be made confidently on the performance of the algorithm on very large networks and large problems. Before the model can be used to

predict the costs for a network of T8 transputers, the values of tf and tc will be required.

The algorithm exhibits good performance with the efficiency falling only to about 0.7 for a matrix of size 256 on a network of 16 transputers. This is encouraging in that even for large numbers of transputers, where communication has a high cost; the algorithm performance does not fall too sharply.

As the number of transputers in the network is increased the speedup for a given matrix size increases to a maximum and then decreases. The number of transputers required to achieve the optimum speedup for the two sections of the algorithm are given by expressions (4.9) and (5.5). These expressions are not the same, but as the matrix reduction has the more dominant cost, the number of transputers used for a given problem size should follow the expression for that section 1. This is clear from Tables 7.5 and 7.6.

The optimum number of transputers for a problem is quite small, thereby restricting the maximum speedup achievable. This is due to the high cost of communication in the algorithm. Better performance may be attained with algorithms based around different network topologies like a mesh. Having a higher connectivity in the network should decrease communication costs by allowing more parallel communication and shorter path lengths and hence increase the optimum number of transputers for the problem.

Due to the different processing speeds of the transputer used for the sequential algorithm and the networked transputers in the RSRE Protonode used for the parallel algorithm, these times have been adjusted to give those timings which would result from running the sequential algorithm on one of the networked transputers.

References

[1] S. H. Abbas, Parallel solution of dense linear equations, *Analele, Universitatti din Timisora, Seria Mathematic a Informatica* **XXXXIX** (2001), 3–12.

[2] S. H. Abbas, On the Cost of Sequential and Parallel Algorithm for Solving Linear System of Equation, *Inter J. Computer. Math* **74** (2000), 391–403.

[3] S. H. Abbas, Complexity of Parallel Block Gauss-Jordan Algorithm, *Inter Journal of Computational and Numerical Analysis and Applications* **4** (2003), 157–166.

[4] M. F. Adam, *A Distributed Memory Unstructured Gauss-Seidel Algorithm for Multigrid Smoothers*, Technical Report, University of California, Berkeley, 2001.

[5] M. F Adam, *A Parallel Maximal Independent Set Algorithm*, In Proceeding 5th copper mountain conference on iterative methods, 1998.

[6] M. F. Adam, *Evaluation of Three Unstructured Multigrid Methods on 3D Finite Element Problems in Solid Mechanics* Technical Report UCB//CSD-00-1103, University of California, Berkeley, 2000.

[7] V. E. Bulgakov and G. Kuhn, High-performance multilevel iterative aggregation solver for large finite-element structural analysis problems, *International Journal for Numerical Methods in Engineering* **38** (1995), 3529–3544.

[8] E. C. H. Chu and J. A. George, Gaussian elimination with partial pivoting and load balancing on a multiprocessor, *Parallel Computing* **5** (1987), 65–74.

[9] M. Cosnard and Y. Robert, Complexity of parallel QR factorization, *J. ACM* **33** (1986), 712–723.

[10] G. J. Davis, *Column LV Factorization with Partial Pivoting on a Hypercube Multiprocessor*, Technical Report ORNL-6219, Mathematical Sciences Section; Oak Ridge National Laboratory, Oak Ridge, TN 37831, 1985.

[11] C. C. Douglas, J. Iskandarani, M. Kowarschik and C. Weiss, Maximizing cache memory usage for multigrid algorithms, In: Multiphase flows and Transport in Porous Media. State of the Art, pp 124–137. Springer, Berlin, 2000.

[12] D. J. Evans *et al.*, *The Double Bordering Algorithm*, V APP III poster session paper, August 1987.

[13] J. Fish, V. Belsky and S. Gomma, Unstructured multigrid method for shells, *International Journal for Numerical Methods in Engineering* **39** (1996), 1181–1197.

[14] V. E. Henson, U. M. Yang A. M. G. Boomer, *A Parallel Algebraic Multigrid Solver and Proconditioner*, Technical Report UCRL- JC139098, Lawrence Livernore National Laboratory, 2000.

[15] C. E. Houstis, E.N. Houstis, J. R. Rice and M. Samartzis, *Benchmarking of Bus Multiprocessor Hardware on Large Scale Scientific Computing*, Dept. of Computer Sciences, Purdue University, 1998.

[16] G. Karpis and V. Kumar, *Parallel Multilevel K-Way Partitioning Scheme for Irregular Graphs*, ACM/IEE Proceeding of SC96: High Performance Networking and Computing, 1996.

[17] G. Li and T. F. Coleman, *A Parallel Triangular Solver for a Hypercube Multiprocessor*, Technical Report 86-787, Department of Computer Science, Cornel University, Ithaca, NY, 1986.

[18] M. Lu and K. Liu, Recursive Relaxation Identification of Linear Multivariable Systems with its Parallel Algorithm, *Information Science* **89** (1996), 211–223.

[19] C. H. Romine and J. M. Ortega, *Parallel Solution of Triangular Systems of Equations*, Applied Mathematics Report RM-86-05, University of Virginia 1986, Parallel Comput.

[20] B. Smith, P. Bjorstad and W. Gropp, *Domain Decomposition*, Cambridge University Press, 1996.

[21] G. W. Stewart, A Parallel implementation of the QR-algorithm, *Parallel Computing* **5** (1987), 187–196.

[22] P. Vanek, J. Mandel and M. Brezina, *Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Problems*, In: 7th Copper Mountain Conference on Multigrid Methods, 1995.