Journal of Computational Mathematics, Vol.14, No.4, 1996, 383–386.

ANALYSING THE EFFICIENCY OF SOLVING DENSE LINEAR EQUATIONS ON DAWNING1000*

X.B. Chi
(*Institute of Software, Chinese Academy of Sciences, Beijing, China*)

Abstract

In this paper, we consider solving dense linear equations on Dawning1000 by using matrix partitioning technique. Based on this partitioning of matrix, we give a parallel block LU decomposition method. The efficiency of solving linear equations by different ways is analysed. The numerical results are given on Dawning1000. By running our parallel program, the best speed up on 32 processors is over 25.

1. Block LU Decomposition Method

The block LU decomposition method is based on the matrix partition technique. Usaully, the matrix A can be partitioned into 1- or 2-directional blocks [?]. In ScaLAPACk, the matrix A is partitioned into 2-directional blocks. For very fast uniprocessor, reducing the communication is more important than reducing a few operations. For this reason and the convenience of FORTRAN program, we partition matrix into column blocks. Assume that $n = mq$. That is,

$$A = \begin{pmatrix} A_0 & A_1 & \dots & A_{q-1} \end{pmatrix}$$

where A_i, $i = 0, \dots, m-1$ are $n \times m$ matrices. For description briefly, we denote the jth processor by P_j. The number of processors is p. In order to get a good load balance, we use column wrap manner to store the partitioned block matrix. That is, A_i is stored in $P_{i \bmod p}$. According to this storage scheme and partitioning method of matrix, the LU decomposition method can be given similarly to that in LINPACK.

Assume that we have a permutation matrix P such that

$$PA = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & 0 \\ L_{10} & I \end{pmatrix} \begin{pmatrix} U_{00} & U_{01} \\ 0 & \tilde{A}_{11} \end{pmatrix}$$

* Received November 8, 1995.

where L_{00} is a unit lower triangular matrix and U_{00} is an upper triangular matrix, then we have:

$$L_{00}U_{00} = A_{00}, \quad L_{10}U_{00} = A_{10}, \quad L_{00}U_{01} = A_{01}, \quad \tilde{A}_{11} = A_{11} - L_{10}U_{01}.$$

From the elimination process of LU decomposition, we know that L_{00}, U_{00} and L_{10} are determined. So we can calculate U_{01} and $\tilde{A}_{11}$ by the above relations. We can use this process to matrix $\tilde{A}_{11}$ too. The block LU decomposition need a lower triangular system solver with multiple right hand side and a matrix multiplication. The serial algorithm is easy, so we don't state it here. The parallel algorithm of block LU decomposition can be described as following:

Algorithm 1:

for $i = 0$ to $q - 1$ do
  if $mynode = i \bmod p$ then
    computing factor L
    if $i \neq q - 1$, sending factor L to $mynode + 1$
  else
    receive factor L from $mynode - 1$
    if $mynode + 1 \neq i \bmod p$, send factor L to $mynode + 1$
  end{if}
  Modifying the rest part of matrix A which includes:
  Solving triangular system with multiple right hand sides
  Doing the matrix multiplication operation
end{for}

In this algorithm, each time it computes one column block of lower triangular matrix L. But the large amount of work for modifying matrix is done parallelly, and the factor L is computed in one processor which does not need communication of messages. The parallel complexity of this algorithm can be given in a similar approach of [**?**]. So it is omitted here. The lower and upper triangular systems solvers can be easily derived from modifying the method of [**?**].

2. Numerical Results

In our test problem, we choose block size to be 8. We use Kernel Mathematics FORTRAN library to construct our program. This library has a lot of functions which are written by Assembly language. Therefore, we can get a very satisfactory actual

speeds of both single and multiple processors on our test problem. The numerical results on Dawning1000 parallel system are listed in the following three tables:

Table 1. Execution Time(s)

Problem Size	1000	2000	4000	8000	10000	15000
nprocs= 1	18.81	125.35	–	–	–	–
nprocs= 2	11.62	70.07	–	–	–	–
nprocs= 4	8.51	44.30	274.26	–	–	–
nprocs= 8	6.29	29.56	164.79	–	–	–
nprocs=16	5.33	22.47	110.58	629.04	1131.64	–
nprocs=32	5.03	18.80	82.50	415.30	716.25	2012.58

Table 2. Actual Speed on Multiprocessor(MFLOPS)

Problem Size	1000	2000	4000	8000	10000	15000
nprocs= 1	35.44	42.55	–	–	–	–
nprocs= 2	57.37	76.11	–	–	–	–
nprocs= 4	78.34	120.39	155.57	–	–	–
nprocs= 8	105.99	180.42	258.92	–	–	–
nprocs=16	125.08	237.35	385.84	542.63	589.12	–
nprocs=32	132.53	283.69	517.17	821.92	930.77	1117.97

Table 3. Speed-Up

Problem Size	1000	2000	4000	8000	10000	15000
nprocs= 2	1.62	1.79	–	–	–	–
nprocs= 4	2.21	2.83	3.51	–	–	–
nprocs= 8	2.99	4.24	5.85	–	–	–
nprocs=16	3.53	5.58	8.72	12.26	13.31	–
nprocs=32	3.74	6.67	11.68	18.57	21.02	25.25

The best float operation speed of single processor is 44.27MFLOPS for solving linear systems, which is achieved at 2500 orders. The speed-up is calculated by the following formular:

$$S_p = \frac{\text{Speed of } p \text{ processors}}{\text{best speed of one processor}}$$

The speed-up values in Table 3 when the orders are great than 2000, are used the above formular.

For nonblock method, the best speed which we can get on uniprocessor is 21MFLOPS. From this simple test result, we can say that the partition technique plays an important role in matrix operation. In this system, if you don't use the methematics library, you can only get 2MFLOPS though you use partitioning technique. Therefore, using basic functions in library is also important to get high efficiency. About using BLAS, it is analysed in [**?**].

In some vector based computers, the library may not play an important role. The partitioning technique is also important for reducing the communication time. This technique is more important for high speed computers than low ones.

References

[1] J.J. Dongarra and D.W. Walker, Software Libraries for Linear Algebra Computations on High Performance Computers, SIAM Review, Vol. 37, 151-180, 1995.

[2] X. B. Chi, Parallel Implementation of Cholesky Decomposition on a Transputer Network, Chinese J. Num. Math. & Appl. Vol 15, 73-80, 1993.

[3] G. Y. Li and T. F. Coleman, A Parallel Triangular Solver for a Hypercube Multiprocessor, TR 86–787, Cornell University, 1986.

[4] J.J. Dongarra, L.S. Duff, D.C. Sorenson, and H.A. van der Vorst, Solving Linear System on Vector and Shared Memory Computers, SIAM Society for Industrial and Applied Mathematics, 1991