CONF-930331--16

LAPACK for Distributed Memory Architectures: The Next Generation*

James Demmel[†] Jack Dongarra[‡§] Robert van de Geijn[¶] David Walker[§]

January 23, 1995

Abstract

We report the progress of an ongoing project that investigates the reusability of LAPACK code for distributed memory MIMD architectures. Major recent revisions include the adoption of a two-dimensional data mapping. This change enhances performance, scalability, and flexibility of the algorithms. Performance results from the Intel iPSC/860 and Intel Touchstone Delta systems are included.

1 Introduction

In this paper, we discuss how the adoption of a two-dimensional data distribution has enhanced the performance, scalability, and flexibility of matrix decomposition algorithms developed as part of an ongoing effort to implement a subset of routines from the LAPACK linear algebra subroutine library on distributed memory architectures. In particular, the affects of these changes on the performance of the QR, LU, and Cholesky decomposition are described. Results from experiments on the Intel iPSC/860 and Intel Touchstone Delta are reported.

The LAPACK project [1] is an effort to update the classical linear algebra software packages LINPACK and EISPACK to allow more efficient use of shared memory and traditional supercomputers. In addition, some of the algorithms have been made more robust. Recently, this package was released to the public domain.

The routines in LINPACK were written in terms of a library of routines known as the Basic Linear Algebra Subprograms (BLAS) [10]. A typical (Level 1) BLAS routine performs a vector operation. To improve performance on machines with hierarchical memories and/or vector processors, the BLAS were extended to include matrix-vector (Level 2 BLAS) and matrix-matrix (Level 3 BLAS) [7, 8] operations. In addition, these routines allow limited encapsulation of parallel operations, mostly on shared memory multiprocessors. LAPACK routines achieve efficiency by maximizing the amount of time spent in higher level BLAS.

In implementing LAPACK routines on distributed memory MIMD architectures, we initially decided to stress minimal change to the algorithms and codes, as well as portability of the final product [2]. The codes were written in FORTRAN77 in the SPMD (Single

*This project was supported in part by the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615, the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract DE-AC05-84OR21400, and Intel Scientific Supercomputing Division.

[†]Computer Science Div. University of California Berkeley, CA 94720

[‡]Computer Science Department, The University of Tennessee, Knoxville, TN 37996.

[§]Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831.

[¶]Department of Computer Sciences, The University of Texas, Austin, TX 78712

"The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. DE-AC05-84OR21400. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes."

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Program Multiple Data) paradigm: all processor nodes of the multicomputer are driven by the same program. The path through the code is determined by the relative position of the node. In addition, we first restricted ourselves to column-oriented mappings of matrices onto the nodes and one-dimensional communication. Portability of the codes was ensured by performing all communication between nodes through a proposed set of Basic Linear Algebra Communication Subprograms (BLACS), which were limited to communicate in one dimension [3].

More recently, both experimental and analytical results were found to indicate that two-dimensional mappings of matrices onto a logical grid of nodes leads to more efficient, scalable, and flexible routines [12, 4]. As a results, a set of two dimensional BLACS had to be defined, allowing communication within rows and columns of the logical grid of nodes [9]. The codes still use FORTRAN77 and the SPMD paradigm. Although this approach leads to considerably more complex code, as we shall see, the benefits are substantial.

This paper is organized as follows: We start by describing the model of parallel computing in Section 2. In Section 3 we describe the two dimensional storage scheme. The algorithms that have been implemented are briefly described in Section 4. Results from experiments on the Intel iPSC/860 and Intel Touchstone Delta are given in Section 5. Concluding remarks are given in the final section.

2 Parallel Computing Model

We assume that our multicomputer consists of p nodes, logically configured as a $P \times Q$ grid and indexed by an ordered pair (I, J), where $0 \leq I < P$ and $0 \leq J < Q$. Each is equipped with CPU and local memory. The nodes are connected by some communication network that allows broadcasting of messages within rows and columns, in addition to point-to-point communication. Messages that arrive over the network are buffered until absorbed by an appropriate call to a receive routine.

In our approach, it is actually not necessary to restrict the model to one process per node. Indeed, whenever we refer to a node, this could just as easily be a process, where many processes could be assigned to a single node.

3 Mapping Matrices to Nodes

There are many ways that matrices can be distributed among nodes. A general class of such mappings can be obtained by partitioning the matrix A like

$$A = \begin{pmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & & \vdots \\ A_{m1} & \cdots & A_{mm} \end{pmatrix}$$

where each subblock A_{ij} is $n_b \times n_b$. These blocks are mapped to nodes by assigning A_{ij} to node $((i-1) \bmod P, (j-1) \bmod Q)$. We refer to this last mapping as the block-torus-wrapped mapping. Another common name for this mapping the block scatter decomposition.

When $P = 1$, this mapping is equivalent of our original column panel-wrapped mapping. In this case, n_b would correspond to the width of the panels.

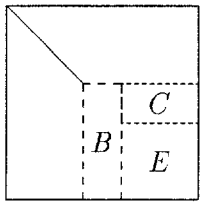
4 Parallel Implementation of the Algorithms

In this section, we outline the parallel implementation of the various algorithms. All are based on right-looking variants of the algorithms. In effect, for the two dimensional

mapping, columns of nodes cooperate to perform the computation that was previously performed by a single node, when panel-wrapped storage was used.

4.1 LU Factorization

Assume the LU factorization has proceeded so that all but the labeled portions of the matrix have been updated:



where $B \in \mathbf{R}^{m \times n_b}$, $C \in \mathbf{R}^{n_b \times (m-n_b)}$, and $E \in \mathbf{R}^{(m-n_b) \times (m-n_b)}$. During the next step, the right-looking algorithm factors panel B, pivoting if necessary. Next, the pivots are applied to the remainder of the matrix. Blocks C and E now become blocks $\bar{C}$ and $\bar{E}$, a triangular solve updates submatrix $\bar{C}$, and a rank n_b update updates submatrix $\bar{E}$. This process continues with the updated matrix.

Turning now to the distributed memory implementation, assume the matrix is block-torus-wrapped onto a grid of nodes of dimension $M \times N$, with block size n_b. The above described process proceeds as follows:

- The column of nodes that holds B collaborating to factor this panel.
- Pivot information is distributed to all other columns of nodes.
- Columns of nodes collaborate to pivot the remainders of the matrix rows.
- Factored panel B is distributed within rows of nodes.
- The row that holds $\bar{C}$ performs the triangular solve, the results of which are distributed within columns of nodes.
- The update of $\bar{E}$ is performed in parallel.

Several implementation details are important to keep in mind:

- Broadcasting the factored panel can be done by sending the panel around the ring of node columns. This has proved to be efficient for column panel-wrapped algorithms, allowing communication and computation to be pipelined and is efficient for block-torus-wrapped approaches for similar reasons [2, 6].
- The broadcast of the updated $\bar{C}$ must be done in a manner that requires the least total time. We have implemented this as a minimum spanning tree broadcast.
- During the factoring of panel B, determining the pivot row and the distribution of this row has been combined to avoid excessive startup overhead.

4.2 Cholesky Factorization

This algorithm proceeds similarly, except that due to symmetry only the lower triangular portion is stored and updated. Moreover, no pivoting is necessary since the matrix is assumed to be positive definite.

4.3 QR Factorization

Again, this algorithm proceeds as the LU factorization, except that it uses Householder transformations to factor the panels. Although no pivoting is necessary, the equivalent of inner products must be accumulated within columns of nodes.

5 Experiments on the Intel Touchstone Delta and iPSC/860

In this section, we report preliminary performance results from implementions of the above algorithms on the Intel Touchstone Delta and iPSC/860. The former resides at Caltech and consists of up to 512 nodes, configured as a grid. The latter resides at Oak Ridge National Laboratory and consists of 128 nodes connected by a hypercube network. Each node consists of an Intel i860 microprocessor. The Delta has 16 Mbytes of memory per node, the iPSC/860 8 Mbytes. All performance results are for the double precision implementations.

The performance attained by various numbers of nodes for the different algorithms are given by Fig. 1-3. Performance is directly related to the ratio between communication and computation. This ratio is most favorable for the QR factorization, and least favorable for Cholesky factorization. This is in part due to the fact that the total number of floating point operations for a given matrix dimension, n, equals $2/3n^3$, $4/3n^3$, and $1/3n^3$ for the LU, QR, and Cholesky factorization, respectively.

We report performance per node attained for the largest problem size as a function of machine size in Fig. 4. In theory, approximately constant efficiency can be maintained when the problem size is increased with the square-root of the number of nodes [4]. This criteria is approximately met by the data used for Fig. 4.

It should be noted that maximum performance per node is around 38 MFLOPS (DP) for matrix-matrix multiplication. However, the reported experiments used a set of BLAS that attained only 30-32 MFLOPS for such operations. This report will be updated with data from a set of BLAS that approaches the 38 MFLOPS performance. Results for the LU factorization using these BLAS have been reported in [4], reaching a peak performance of 14 GFLOPS on the Delta.

The data for the LU factorization reported in this paper were obtained using a faster version of the operating system, allowing slightly faster communication.

6 Conclusion

Our research shows that good performance can be attained for these algorithms on architectures like the Intel machines. Portability is enhanced through the use of the BLACS. The two-dimensional data distribution allows more flexibility, in addition to better performance and scalability (for details see [4]).

Acknowledgments

This research was performed in part using the Intel Touchstone Delta System operated by the California Institute of Technology on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by the California Institute of Technology and Intel Supercomputer Systems Division. In addition, the Intel iPSC/860 at Oak Ridge National Laboratory was utilized.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. LAPACK: A portable linear algebra library for

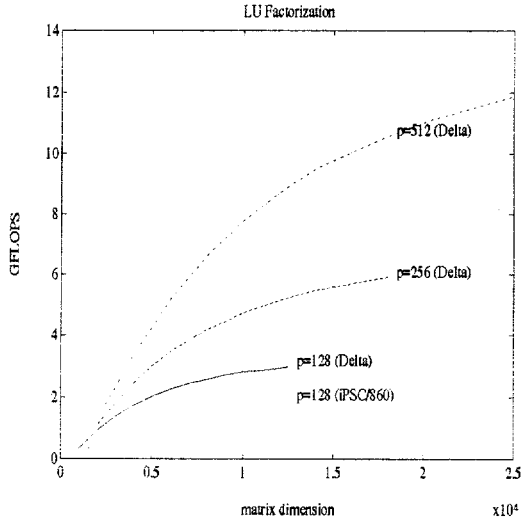


FIG. 1. *Performance attained by the LU factorization.*

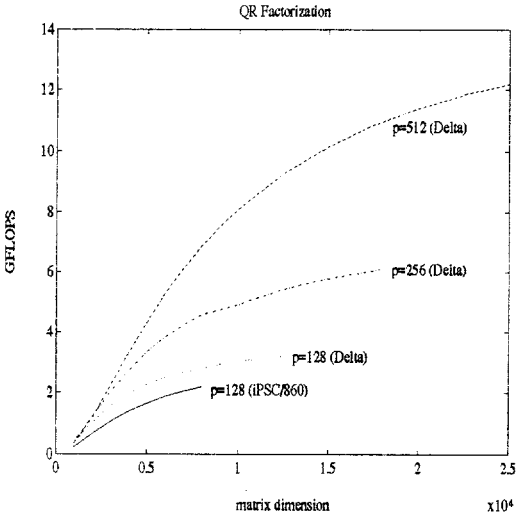


FIG. 2. *Performance attained by the QR factorization.*

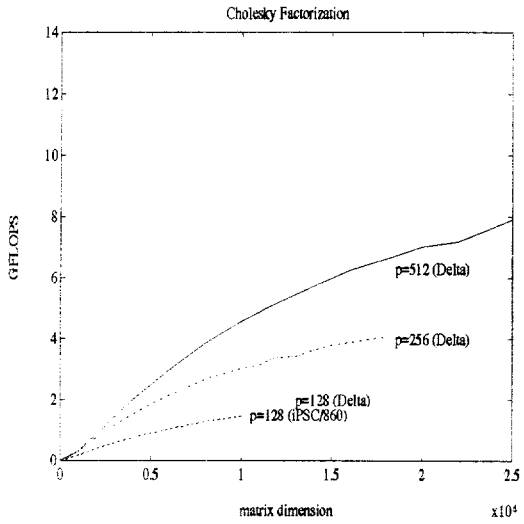


FIG. 3. *Performance attained by the Cholesky factorization.*

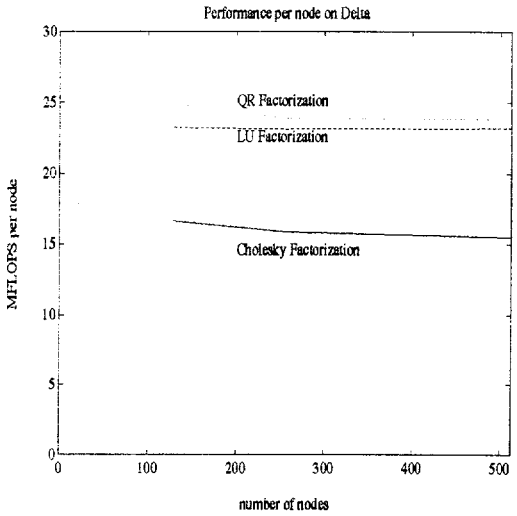


FIG. 4. *Performance per node on various machine sizes for the largest matrix dimension* ($n^2/p \approx$ constant).

high-performance computers. In *Proceedings Supercomputing '90*, pages 2–11. IEEE Computer Society Press, Los Alamitos, California, 1990.

[2] E. Anderson, A. Benzoni, J. Dongarra, S. Moulton, S. Ostrouchov, B. Tourancheau, and R. van de Geijn. LAPACK for distributed memory architectures: Progress report. In *Parallel Processing for Scientific Computing, Fifth SIAM Conference*. SIAM, 1991.

[3] E. Anderson, A. Benzoni, J. Dongarra, S. Moulton, S. Ostrouchov, B. Tourancheau, and R. van de Geijn. Basic Linear Algebra Communication Subprograms. In *Sixth Distributed Memory Computing Conference Proceedings*, pages 287–290. IEEE Computer Society Press, 1991.

[4] J. Dongarra, R. van de Geijn, and D. Walker. A look at scalable dense linear algebra libraries. In *Proceedings of SHPCC92*, April, 1992.

[5] J. J. Dongarra, I. S. Duff, and D. C. Sorensen H. A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers*. SIAM, Philadelphia, PA, 1990.

[6] Jack Dongarra and Susan Ostrouchov. LAPACK block factorization algorithms on the Intel iPSC/860. LAPACK Working Note 24, Technical Report CS-90-115, University of Tennessee, Oct. 1990.

[7] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.

[8] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 14(1):1–17, March 1988.

[9] J.J. Dongarra and R.A. van de Geijn. Two dimensional basic linear algebra subprograms. LAPACK Working Note 37, Technical Report CS-91-138, University of Tennessee, 1991.

[10] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Soft.*, 5(3):308–323, Sept. 1979.

[11] S.L. Lillevik. The Touchstone 30 Gigaflop DELTA Prototype. In *Sixth Distributed Memory Computing Conference Proceedings*, pages 671–677. IEEE Computer Society Press, 1991.

[12] R.A. van de Geijn. Massively parallel LINPACK benchmark on the Intel Touchstone Delta and iPSC/860 systems. In *1991 Annual Users' Conference Proceedings*. Intel Supercomputer Users' Group, Dallas, TX, 1991.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.