The design of a parallel dense linear algebra software library: Reduction to Hessenberg, tridiagonal, and bidiagonal form*

Jaeyoung Choi, a Jack J. Dongarra a,b and David W. Walker b

aDepartment of Computer Science, University of Tennessee at Knoxville, 107 Ayres Hall, Knoxville, TN 37996-1301, USA
bMathematical Sciences Section, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 6012, Oak Ridge, TN 37831-6367, USA

Received 19 January 1995
Communicated by C. Brezinski

This paper discusses issues in the design of ScaLAPACK, a software library for performing dense linear algebra computations on distributed memory concurrent computers. These issues are illustrated using the ScaLAPACK routines for reducing matrices to Hessenberg, tridiagonal, and bidiagonal forms. These routines are important in the solution of eigenproblems. The paper focuses on how building blocks are used to create higher-level library routines. Results are presented that demonstrate the scalability of the reduction routines. The most commonly-used building blocks used in ScaLAPACK are the sequencing BLAS, the parallel BLAS (PBLAS) and the Basic Linear Algebra Communication Subprograms (BLACS). Each of the matrix reduction algorithms consists of a series of steps in each of which one block column (or panel), and/or block row, of the matrix is reduced, followed by an update of the portion of the matrix that has not been factorized so far. This latter phase is performed using Level 3 PBLAS operations and contains the bulk of the computation. However, the panel reduction phase involves a significant amount of communication, and is important in determining the scalability of the algorithm. The simplest way to parallelize the panel reduction phase is to replace the BLAS routines appearing in the LAPACK routine (mostly matrix-vector and matrix-matrix multiplications) with the corresponding PBLAS routines. However, in some cases it is possible to reduce communication startup costs by performing the communication necessary for consecutive BLAS operations in a single communication using a BLACS call. Thus, there is a tradeoff between efficiency and software engineering considerations, such as ease of programming and simplicity of code.

1. Introduction

This paper addresses issues in the design and implementation of ScaLAPACK, a

* Research was supported in part by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy, by the Defense Advanced Research Projects Agency under contract DAAL03-91-C-0047, administered by the Army Research Office, and in part by the Center for Research on Parallel Computing.

© J.C. Baltzer AG, Science Publishers
software library for performing dense linear algebra computations on distributed memory concurrent computers. Upon completion, ScaLAPACK ("Scalable LAPACK") will make available on distributed memory machines the same set of library routines that LAPACK [1,2] provides for vector and shared memory architectures.

A set of Basic Linear Algebra Subprograms (Level 1, 2, and 3 BLAS) [8,11,20] is available as a highly efficient machine-specific implementation on many modern high-performance computers. They provide high performance with portability and are used as the building blocks of a number of applications, including LAPACK. The Basic Linear Algebra Communication Subprograms (BLACS) [10] comprise a package that provides ease-of-use and portability for message-passing in parallel linear algebra applications. The Parallel BLAS (PBLAS), which provide a simplified interface around the Parallel Block BLAS (PB- BLAS) [7], are intermediate level routines based on the sequential BLAS and the BLACS. The PBLAS provide all the functionality supported by parallel versions of the Level 1, 2, and 3 BLAS on a restricted class of matrices having a block cyclic data distribution. The ScaLAPACK routines are built using the sequential BLAS, the BLACS, and the PBLAS modules. ScaLAPACK can be ported with minimal code modification to any machine on which the BLAS and the BLACS are available.

Of particular interest in this paper is the tradeoff between performance and modular algorithm design. This tradeoff will be illustrated using routines that use Householder transformations to reduce a real general matrix to Hessenberg or bidiagonal form, and a symmetric matrix to tridiagonal form. The reduction of a matrix to Hessenberg form is an important computational component in the unsymmetric eigenvalue problem. The reduction to tridiagonal form plays a similar role in the symmetric eigenvalue problem. Reduction to bidiagonal form is important in evaluating the singular value decomposition (SVD) of a matrix, which in turn is used in the least-squares solution of overdetermined systems of linear equations.

Currently ScaLAPACK also includes LU, QR, and Cholesky factorization routines with their solvers. The implementation details, performance, and scalability of the ScaLAPACK factorization routines are presented in a separate paper [4].

The design philosophy of the ScaLAPACK library is addressed in section 2. In section 3, we introduce the block equations of the reduction routines and describe the ScaLAPACK reduction routines by comparing them with the corresponding LAPACK routines. Section 4 presents performance results and scalability of the algorithms on the Intel family of computers: the iPSC/860, the Touchstone Delta, and the Paragon. In section 5, conclusions and future work are presented.

2. Design philosophy

In ScaLAPACK, algorithms are presented in terms of *processes* rather than the