2.1. Structure and Functionality of the Scalable Linear Algebra PACKage

Much of the work in developing linear algebra software for advanced-architecture computers is motivated by the need to solve large problems on the fastest computers available. In this chapter, we'll discuss the development of standards for linear algebra software, the building blocks for software libraries, and aspects of algorithm design as influenced by the opportunities for parallel implementation.

The **Linear Algebra PACKage (LAPACK)** for many years is designed to be highly efficient on vector processors, high-performance "superscalar" workstations, and shared-memory multiprocessors. LAPACK can also be used satisfactorily on all types of scalar machines (PCs, workstations, and mainframes). However, LAPACK in its present form is less likely to give good performance on other types of parallel architectures, for example, massively parallel Single Instruction Multiple Data (SIMD) machines, or Multiple Instruction Multiple Data (MIMD) distributed-memory machines. The **Scalable Linear Algebra PACKage,** or **ScaLAPACK** effort was intended to adapt LAPACK to these new architectures. By creating the ScaLAPACK software library, was extended the LAPACK library to scalable MIMD, distributed-memory, concurrent computers. For such machines, the memory hierarchy includes the off-processor memory of other processors, in addition to the hierarchy of registers, cache, and local memory on each processor. *So the ScaLAPACK is a library of linear algebra routines for high-performance computing.* These routines can be used to solve Linear Least Squares (LLS) problems, singular value and eigenvalues problems, and systems of linear equations. ScaLAPACK was derived from LAPACK and to exploit parallelism, a subset of the LAPACK routines were rewritten for use on distributed-memory processor computers.

Like LAPACK, the ScaLAPACK routines are based on *block-partitioned algorithms* in order to minimize the frequency of data movement between different levels of the memory hierarchy. The fundamental building blocks of the ScaLAPACK library are distributed-memory versions of the Level-2 and Level-3 of the **Basic Linear Algebra Subprograms** or **BLAS**, and a set of **Basic Linear Algebra Communication Subprograms (BLACS)** for communication tasks that arise frequently in parallel linear algebra computations. In the ScaLAPACK routines, the majority of interprocessor communication occurs within the **Parallel Basic Linear Algebra Subprograms** (PBLAS), so the source code of the top software layer of ScaLAPACK looks similar to that of LAPACK.

ScaLAPACK contains:

- **driver routines** for solving standard types of problems**,**
- **computational routines to perform a distinct computational task,**
- **auxiliary routines to perform a certain subtask or common low-level computation.**

Each driver routine typically calls a sequence of computational routines. Taken as a whole, the computational routines can perform a wider range of tasks than are covered by the driver routines. Many of the auxiliary routines may be of use to numerical analysts or software developers, so we have documented the Fortran source for these routines with the same level of detail used for the ScaLAPACK computational routines and driver routines.

As mentioned before, **LAPACK** is a collection of routines for solving **linear systems, least squares problems, eigen problems, and singular problems**. High performance is attained by using algorithms that do most of their work in calls to the BLAS, with an emphasis on matrix-matrix multiplication. The LAPACK routines are written as a single thread of execution. LAPACK can accommodate shared-memory machines, provided parallel BLAS are available (in other words, the only parallelism is implicit in calls to BLAS).

The **BLAS** include subroutines for common linear algebra computations such **as dot-products, matrix-vector multiplication, and matrix-matrix multiplication**. An important aim of the BLAS is to provide a portability layer for computation.

PBLAS, a parallel set of BLAS, called perform message-passing and whose interface is as similar to the BLAS as possible. This permitted the ScaLAPACK code to be quite similar, and sometimes nearly identical, to the analogous LAPACK code. We hope that the PBLAS will provide a distributed memory standard , just as the BLAS have provided a shared memory standard. This would simplify and encourage the development of high performance and portable parallel numerical software, as well as providing manufacturers with a small set of routines to be optimized.

The **BLACS** are a message-passing library designed for linear algebra. The computational model consists of a one- or two-dimensional process grid, where each process stores pieces of the matrices and vectors. The BLACS include s**ynchronous send/receive routines** to communicate a matrix or submatrix from one process to another, **to broadcast submatrices** to many processes, or **to compute global reductions** (sums, maxima and minima). There are also **routines to construct, change, or query the process grid**. Since several ScaLAPACK algorithms require broadcasts or reductions among different subsets of processes, the BLACS permit a process to be a member of several overlapping or disjoint process grids, each one labeled by a **context**. Some message-passing systems, such as MPI, also include this context concept; MPI calls this a **communicator**. The BLACS provide facilities for safe inter-operation of system contexts and BLACS contexts. An important aim of the BLACS is to provide a portable, linear algebra specific layer for communication.

The ScaLAPACK strategy for combining efficiency with portability is to construct the software so that as much as possible of the computation is performed by calls to the Parallel Basic Linear Algebra Subprograms (PBLAS). The PBLAS perform global computation by relying [bazindu-se pe] on the Basic Linear Algebra Subprograms (BLAS) for local computation and the Basic Linear Algebra Communication Subprograms (BLACS) for communication. The efficiency of ScaLAPACK software depends on the **use of block-partitioned algorithms** and on efficient implementations of the BLAS and the BLACS being provided by computer vendors (and others) for their machines. Thus, the BLAS and the BLACS form a low-level interface between ScaLAPACK software and different machine architectures. Above this level, all of the ScaLAPACK software is portable.

In Figure 1 we present the **hierarchy structure of the software ScaLAPACK**.

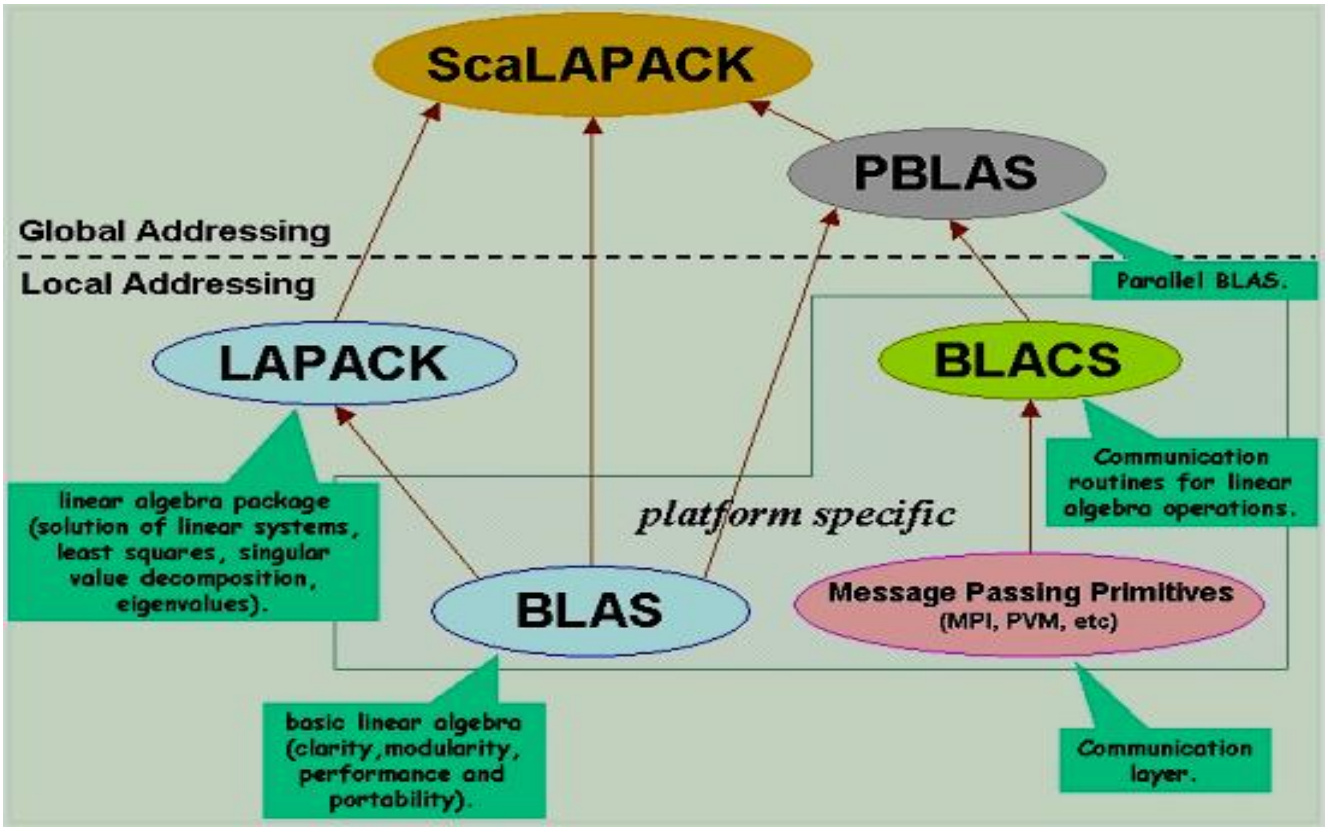

Figure 1.

Data Types and Precision.

ScaLAPACK provides the same range of functionality for **real** and **complex** data, with a few exceptions. Matching routines for real and complex data have been coded to maintain a close correspondence between the two, wherever possible. However, there are cases where the corresponding complex version calling sequence has more arguments than the real version. All routines in ScaLAPACK are provided in both **single**- and **double**-precision versions.

Step-by-Step Procedure for using a ScaLAPACK routine.

Four basic steps are required to call a ScaLAPACK routine.

- Initialize the process grid.
- Distribute the matrix on the process grid.
- Call ScaLAPACK routine.
- Release the process grid.

These steps can be described in details by means of ScaLAPACK components as follows:

1) Write a scaled-down program using the equivalent LAPACK routine.
   - Good for familiarizing user with the proper routine name and arguments. Parallel routine name and arguments will be similar,
   - Good for syntax/logical debugging.
2) Initialize the BLACS library for its use in the program.
3) Create and use the BLACS processor grid.
4) Distribute pieces of each global array over the processors in the grid.
   - User does this by creating an *array descriptor vector* for each global array,
   - Global array elements mapped in a 2-D blocked-cyclic manner onto the

processor grid.

5) Have each processor initialize its local array with the correct values of the pieces of the global array it owns.
6) Call the ScaLAPACK routine.
7) Confirm/use the output of the ScaLAPACK routine.
8) Release the processor grid and exit the BLACS library.

Thus, the main purpose of this chapter is to describe ScaLAPACK Routines so that the user can realize the steps 1)-8) for parallel program elaboration to solve some practical problems of the linear systems simulation.