LAPACK Working Note 100
A Proposal for a Set of Parallel Basic Linear Algebra Subprograms *

J. Choi[†], J. Dongarra[‡], S. Ostrouchov[§], A. Petitet[§], D. Walker[¶], and R. C. Whaley[§]

May, 1995

Abstract

This paper describes a proposal for a set of Parallel Basic Linear Algebra Subprograms (PBLAS). The PBLAS are targeted at distributed vector-vector, matrix-vector and matrix-matrix operations with the aim of simplifying the parallelization of linear algebra codes, especially when implemented on top of the sequential BLAS.

At first glance, because of the apparent simplicity of its sequential counterpart as well as the regularity of the data structures involved in dense linear algebra computations, implementing an equivalent set of parallel routines in terms of portability, efficiency, and ease-of-use seems relatively simple to achieve.

However, when these routines are actually coded, the problem becomes much more complex due to difficulties which do not occur in serial computing. First, there are many different parallel computer architectures available. In view of this fact, it is natural to choose a virtual machine topology that is convenient for dense linear algebra computations and map the virtual machine onto existing topologies. Second, the selected data distribution scheme must ensure good load-balance to guarantee performance and scalability. Finally, for ease-of-use and software reusability reasons, the interface of the top-level routines must closely resemble the sequential BLAS interface yet still be flexible enough to take advantage of efficient parallel algorithmic techniques such as computation and communication overlapping and pipelining.

This paper presents a reasonable set of adoptable solutions to successfully design and implement the Parallel Basic Linear Algebra Subprograms. These subprograms can in turn be used to develop parallel libraries such as ScaLAPACK for a large variety of distributed memory MIMD computers.

*This work was supported in part by the National Science Foundation Grant No. ASC-9005933; by the Defense Advanced Research Projects Agency under contract DAAL03-91-C-0047, administered by the Army Research Office; by the Office of Scientific Computing, U.S. Department of Energy, under Contract DE-AC05-84OR21400; and by the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615.

†School of Computing, Soogsil University, Seoul 156-743, Korea. The author's research was performed at the Department of Computer Science of University of Tennessee and Oak Ridge National Laboratory

‡Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, and Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831

§Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301

¶Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831

Contents

1 Introduction

In 1987 Dongarra, Du Croz, Duff and Hammarling wrote an article in the *ACM Trans. Math. Soft.* (Vol. 16, no. 1, page 1) defining and proposing a set of Level 3 Basic Linear Algebra Subprograms. That proposal logically concluded a period of reflection and discussion among the mathematical software community [12, 21, 24] to define a set of routines that would find wide application in software for numerical linear algebra and provide a useful tool for implementors and users. Because these subprograms and their predecessors – the Levels 1 and 2 BLAS – are an aid to clarity, portability, modularity and maintenance of software, they have been embraced by the community and have become a *de facto* standard for elementary linear algebra computations [11].

Many of the frequently used algorithms of numerical linear algebra can be implemented so that a majority of the computation is performed by calls to the Level 2 and Level 3 BLAS. By relying on these basic kernels, it is possible to develop portable and efficient software across a wide range of architectures, with emphasis on workstations, vector-processors and shared-memory computers, as has been done in LAPACK [2].

As opposed to shared-memory systems, distributed-memory computers differ significantly from the software point of view. The underlying interconnection network as well as the vendor supplied communication library are usually machine specific. The ongoing Message Passing Interface (MPI) standardization effort [17] will undoubtly be of great benefit to the user community. Nevertheless, a large variety of distributed-memory systems still exists and this motivated the development of a set of portable communication subprograms well suited for linear algebra computations: the Basic Linear Algebra Communication Subprograms (BLACS) [14, 27]. In addition to defining a portable interface the BLACS also provide the correct level of abstraction. They allow the software writer to focus on performing message passing on subsections of matrices rather than at low level byte transfers.

There has been much interest recently in developing parallel versions of the BLAS for distributed memory computers [1, 3, 15, 16]. Some of this research proposed parallelizing the BLAS, and some implemented a few important BLAS routines, such as matrix-matrix multiplication. Almost ten years after the very successful BLAS were proposed, we are in a position to define and implement a set of Basic Linear Algebra Subprograms for distributed-memory computers with similar functionality as their sequential predecessors. The proposed set of routines that constitute the Parallel Basic Linear Algebra Subprograms results from the adaptation to distributed memory computers and reuse of the design decisions made for the BLAS. The local computations within a process are performed by the BLAS, while the communication operations are handled by the BLACS.

In an effort to simplify the parallelization of serial codes implemented on top of the BLAS, the PBLAS proposed here are targeted at vector-vector, matrix-vector and matrix-matrix operations. The last section illustrates how some common algorithms can be implemented by calls to the proposed routines. There is certainly considerable evidence for the efficiency of such algorithms on various machines [18]. Such implementations are portable and efficient across a wide variety of distributed memory MIMD computers, ranging from a heterogeneous network of workstations to a statically connected set of identical processors, provided that efficient machine-specific BLAS and BLACS are available.

The scope of this proposal is limited. First, the set of routines described in this paper constitutes an extended proper subset of the BLAS. For instance, this proposal does not contain vector rotation routines or dedicated subprograms for banded or packed matrices. A matrix transposition routine has been added to the Level 3 subprograms since this operation is much more complex to perform and implement on distributed-memory computers. Second, this proposal does not include routines for matrix factorizations or reductions; these are covered by the ScaLAPACK (Scalable Linear Algebra PACKage) project [6, 7]. A reference implementation version of the PBLAS is available on netlib (http://www.netlib.org). Vendors can then supply system optimized versions of the BLAS, the BLACS and eventually the PBLAS. It is our hope that this proposal will initiate discussions among the computer science community so that this project will best reflect its needs.

This proposal is intended primarily for software developers and to a lesser extent for experienced applications programmers. The details of this proposal are concerned with defining a set of subroutines for use in FORTRAN 77 and C programs. However, the essential features of this standard should be easily adaptable to other programming languages. We have attempted to pave the way for such a future evolution by respecting the driving concepts of the HPF [23] and MPI [17] projects.

2 Scope of the PBLAS

The design of the software is as consistent as possible with that of the BLAS; thus, the experienced linear algebra programmer will have the same basic tools available in both the sequential and parallel programming worlds.

In real arithmetic the operations for the PBLAS have the following form:

- Level 1 - Vector-vector operations
  - $x \leftrightarrow y$
  - $x \leftarrow \alpha x$
  - $y \leftarrow x$
  - $y \leftarrow \alpha x + y$
  - $dot \leftarrow x^T y$
  - $nrm2 \leftarrow \|x\|_2$
  - $asum \leftarrow \|re(x)\|_1 + \|im(x)\|_1$
  - Index and value of the first maximal element in absolute value of a vector.
- Level 2 - Matrix-vector operations
  - Matrix-vector products
    - $y \leftarrow \alpha A x + \beta y$
    - $y \leftarrow \alpha A^T x + \beta y$
  - Rank-1 update of a general matrix
    - $A \leftarrow \alpha x y^T + A$

- Rank-1 and rank-2 updates of a symmetric matrix
  - $A \leftarrow \alpha x x^T + A$
  - $A \leftarrow \alpha x y^T + \alpha y x^T + A$
- Multiplication by a triangular matrix
  - $x \leftarrow Tx$
  - $x \leftarrow T^T x$
- Solving a triangular system of equations
  - $x \leftarrow T^{-1} x$
  - $x \leftarrow T^{-T} x$

- Level 3 - Matrix-matrix operations
  - Matrix-matrix products
    - $C \leftarrow \alpha AB + \beta C$
    - $C \leftarrow \alpha A^T B + \beta C$
    - $C \leftarrow \alpha AB^T + \beta C$
    - $C \leftarrow \alpha A^T B^T + \beta C$
  - Rank-k and rank-$2k$ updates of a symmetric matrix
    - $C \leftarrow \alpha AA^T + \beta C$
    - $C \leftarrow \alpha A^T A + \beta C$
    - $C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$
    - $C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$
  - Multiplication by a triangular matrix
    - $B \leftarrow \alpha TB$
    - $B \leftarrow \alpha T^T B$
    - $B \leftarrow \alpha BT$
    - $B \leftarrow \alpha BT^T$
  - Solving multiple triangular systems of equations
    - $B \leftarrow \alpha T^{-1} B$
    - $B \leftarrow \alpha T^{-T} B$
    - $B \leftarrow \alpha BT^{-1}$
    - $B \leftarrow \alpha BT^{-T}$
  - Matrix transposition
    - $C \leftarrow \beta C + \alpha A^T$

Here α and β are scalars, x and y are vectors, A, B and C are rectangular matrices (in some cases square and symmetric), and T is an upper or lower triangular matrix (and nonsingular for the triangular solves).

Analogous operations are proposed in complex arithmetic. Conjugate transposition is specified as well as simple transposition. Additional operations are provided for scaling a complex vector by a real scalar and updates of a Hermitian matrix as follows:

* $x \leftarrow \alpha x$
* $A \leftarrow \alpha x x^H + A$
* $C \leftarrow \alpha A A^H + \beta C$
* $C \leftarrow \alpha A^H A + \beta C$

with α and β real for the vector-vector and matrix-matrix operations, and

* $A \leftarrow \alpha x y^H + y(\alpha x)^H + A$
* $C \leftarrow \alpha A B^H + \bar{\alpha} B A^H + \beta C$
* $C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C$

with β real.

3 Conventions of the PBLAS

3.1 Naming Conventions

The name of a PBLAS routine follows the conventions of the BLAS with the exception that the first character in the name is always a **'P'**, which stands for Parallel. The second character (corresponding to the first character in BLAS names) denotes the FORTRAN data type of the matrix or vector as follows:

S REAL

D DOUBLE PRECISION

C COMPLEX

Z COMPLEX*16 or DOUBLE COMPLEX (if available)

The last characters in the name of the Level 1 routines are abbreviations of the performed operations as indicated in Table 1. For example `PSCOPY` is the single precision real vector-vector copy routine name. The third and fourth characters in the name of the Levels 2 and 3 routines refer to the kind of matrix involved as follows:

GE All matrix operands are GEneral rectangular;

HE One of the matrix operands is HErmitian;

SY One of the matrix operands is SYmmetric;

TR One of the matrix operands is TRiangular.

Name	Function	Prefixes
P□SWAP	Swap x and y	S, C, D, Z
P□SCAL	Constant times a vector	S, C, D, Z, CS, ZD
P□COPY	Copy x into y	S, C, D, Z
P□AXPY	Constant times a vector plus a vector	S, C, D, Z
P□DOT	Dot product	S, D
P□DOTU	Dot product	C, Z
P□DOTC	Dot product	C, Z
P□NRM2	2-norm (Euclidean length)	S, D, SC, DZ
P□ASUM	Sum of absolute values (*)	S, D, SC, DZ
P□AMAX	Index and value of element having maximum absolute value (*)	S, C, D, Z

(*) For complex components $z_j = x_j + iy_j$ these subprograms compute $|x_j| + |y_j|$ instead of $(x_j^2 + y_j^2)^{1/2}$.

Table 1: Summary of proposed Level 1 PBLAS routines

The fifth and sixth characters in the name of the Levels 2 and 3 routines denote the type of operation as follows:

MM Matrix-matrix product;

MV Matrix-vector product;

R Rank-1 update of a matrix;

R2 Rank-2 update of a matrix;

RK Rank-k update of a symmetric or Hermitian matrix;

R2K Rank-$2k$ update of a symmetric or Hermitian matrix;

SM Solves a system of linear equations for a matrix of right-hand sides;

SV Solves a system of linear equations for a right-hand side vector.

The suggested combinations are indicated in Table 2. Note, however, that rank-k updates of general matrices are provided by the `GEMM` routines. In the first column, under *real* the second character `S` may be replaced by `D`. In the second column, under *complex*, the second character `C` may be replaced by `Z`. See appendix C for the complete subroutine calling sequences.

In our model implementation, however, the matrix transposition routine is an exception and is called `P□TRAN□`.

The collection of routines can be thought of as being divided into four separate parts, *real, double precision, complex and complex*16.* These routines can be written in C or FORTRAN 90 for

Real	Complex	MM	MV	R	R2	RK	R2K	SM	SV
PSGE	PCGE	*	*	*	*				
PSSY	PCSY	*	*	*	*	*	*		
	PCHE	*	*	*	*	*	*		
PSTR	PCTR	*	*					*	*
Matrix transposition: PSTRAN, PCTRANU and PCTRANC.									

Table 2: Summary of proposed Level 2 and 3 PBLAS routines

example; their implementation takes advantage of dynamic memory management features present in these programming languages. However, as we will see later, the local storage convention of the distributed matrix operands in every process's memory is assumed to be FORTRAN like, i.e., "column major" as it is specified for the BLAS. Thus, it is possible to rely on the BLAS to perform the local computations within a process.

3.2 Storage Conventions

The current model implementation of the PBLAS assumes the matrix operands to be distributed according to the block-cyclic decomposition scheme. This allows the routines to achieve scalability, well balanced computations and to minimize synchronization costs. It is not the object of this paper to describe in detail the data mapping onto the processes, for further details see [7, 13]. Let us simply say that the set of processes is mapped to a virtual mesh, where every process is naturally identified by its coordinates in this $P \times Q$ grid. This virtual machine is in fact part of a larger object defined by the BLACS and called a context [14].

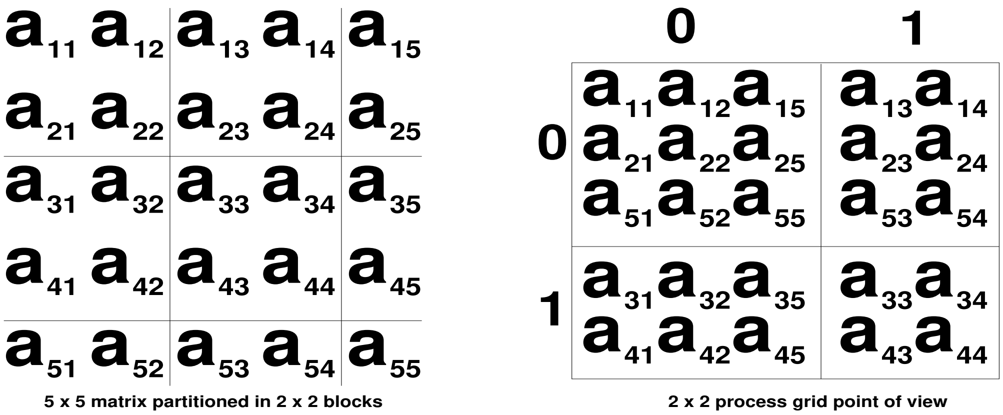


Figure 1: A 5×5 matrix decomposed into 2×2 blocks mapped onto a 2×2 process grid.

An M_ by N_ matrix operand is first decomposed into MB_ by NB_ blocks starting at its upper left corner. These blocks are then uniformly distributed across the process mesh. Thus every process owns a collection of blocks, which are locally and contiguously stored in a two dimensional "column major" array. We present in Fig. 1 the mapping of a 5 $\times$ 5 matrix partitioned into 2 $\times$ 2 blocks mapped onto a 2×2 process grid, i.e M_=N_=5 and MB_=NB_= 2. The local entries of every matrix column are contiguously stored in the processes' memories.

It follows that a general M_ by N_ distributed matrix is defined by its dimensions, the size of the elementary MB_ by NB_ block used for its decomposition, the coordinates of the process having in its local memory the first matrix entry {RSRC_,CSRC_}, and the BLACS context (CTXT_) in which this matrix is defined. Finally, a local leading dimension LLD_ is associated with the local memory address pointing to the data structure used for the local storage of this distributed matrix. In Fig. 1, we choose for illustration purposes RSRC_=CSRC_=0. In addition, the local arrays in process row 0 must have a leading dimension LLD_ greater than or equal to 3, and greater than or equal to 2 in the process row 1.

These pieces of information are grouped together into a single 8 element integer array, called the descriptor, DESC_. Such a descriptor is associated with each distributed matrix. The entries of the descriptor uniquely determine the mapping of the matrix entries onto the local processes' memories. Moreover, with the exception of the local leading dimension, the descriptor entries are global values characterizing the distributed matrix operand. Since vectors may be seen as a special case of distributed matrices or proper submatrices, the larger scheme just defined encompasses their description as well.

For distributed symmetric and Hermitian matrices, only the upper (UPLO='U') triangle or the lower (UPLO='L') triangle is stored. For triangular distributed matrices, the argument UPLO serves to define whether the matrix is upper (UPLO='U') or lower (UPLO='L') triangular.

For a distributed Hermitian matrix the imaginary parts of the diagonal elements are zero and thus the imaginary parts of the corresponding FORTRAN or C local arrays need not be set, but are assumed to be zero. In the P□HER and P□HER2 routines, these imaginary parts will be set to zero on return, except when α is equal to zero, in which case the routines exit immediately. Similarly, in the P□HERK and P□HER2K routines the imaginary parts of the diagonal elements will also be set to zero on return, except when β is equal to one and α or K is equal to zero, in which case the routines exit immediately.

3.3 Argument Conventions

The order of the arguments of a PBLAS routine is as follows:

1. Arguments specifying matrix options
2. Arguments defining the sizes of the distributed matrix or vector operands
3. Input-Output scalars
4. Description of the input distributed vector or matrix operands
5. Input scalar (associated with the input-output distributed matrix or vector operand)
6. Description of the input-output distributed vector or matrix operands

Note that every category is not present in each of the routines. The arguments that specify options are character arguments with the names `SIDE`, `TRANS`, `TRANSA`, `TRANSB`, `UPLO` and `DIAG`.

`SIDE` is used by the routines as follows:

Value	Meaning
'L'	Multiply general distributed matrix by symmetric or triangular distributed matrix on the left.
'R'	Multiply general distributed matrix by symmetric or triangular distributed matrix on the right.

`TRANS`, `TRANSA` and `TRANSB` are used by the routines as follows:

Value	Meaning
'N'	Operate with the distributed matrix.
'T'	Operate with the transpose of the distributed matrix.
'C'	Operate with the conjugate transpose of the distributed matrix.

In the real case the values 'T' and 'C' have the same meaning, and in the complex case the value 'T' is not allowed.

`UPLO` is used by the Hermitian, symmetric, and triangular distributed matrix routines to specify whether the upper or lower triangle is being referenced as follows:

Value	Meaning
'U'	Upper triangle.
'L'	Lower triangle.

`DIAG` is used by the triangular distributed matrix routines to specify whether or not the distributed matrix is unit triangular, as follows:

Value	Meaning
'U'	Unit triangular.
'R'	Non-unit triangular.

When `DIAG` is supplied as 'U' the diagonal elements are not referenced.

Thus, these arguments have similar values and meanings as for the BLAS; `TRANSA` and `TRANSB` have the same values and meanings as `TRANS`, where `TRANSA` and `TRANSB` apply to the distributed matrix operands `A` and `B` respectively. We recommend that the equivalent lower case characters be accepted with the same meaning.

The distributed submatrix operands of the Level 3 PBLAS are determined by the arguments `M`, `N` and `K`, which specify their size. These numbers may differ from the two first entries of the descriptor (`M_` and `N_`), which specifies the size of the distributed matrix containing the submatrix operand. Also required are the global starting indices `IA`, `JA`, `IB`, `JB`, `IC` and `JC`. It is permissible to call a routine with M or N equal to zero, in which case the routine exits immediately without referencing its distributed matrix arguments. If M and N are greater than zero, but K is equal to zero, the operation reduces to C(IC:*,JC:*)$\leftarrow$ βC(IC:*,JC:*) (this applies to the GEMM, SYRK, SYR2K, HERK and HER2K routines). The input-output distributed submatrix (B(IB:*,JB:*) for the TR-routines, C(IC:*,JC:*) otherwise) is always M $\times$ N if rectangular, or N $\times$ N if square.

The description of the distributed matrix operands consists of

- a pointer in every process to the local array (A, B or C) containing the local pieces of the corresponding distributed matrix,
- the global starting indices in row column order { (IA, JA), (IB, JB), (IC, JC) },
- the descriptor of the distributed matrix as declared in the calling (sub)program (DESCA, DESCB or DESCC).

The description of a distributed vector operand is similar to the description of a distributed matrix (X, IX, JX, DESCX) followed by a global increment INCX, which allows the selection of a matrix row or a matrix column as a vector operand. Only two increment values are currently supported by our model implementation, namely 1 to select a matrix column and DESCX(1) (i.e INCX=MX) specifying a matrix row.

The input scalars always have the dummy argument names ALPHA and BETA. Output scalars are only present in the Level 1 PBLAS and are called AMAX, ASUM, DOT, INDX and NORM2.

We use the description of two distributed matrix operands X and Y to describe the invalid values of the arguments:

- Any value of the character arguments SIDE, TRANS, TRANSA, TRANSB, UPLO, or DIAG, whose meaning is not specified,
- M $<$ 0 or N $<$ 0 or K $<$ 0,
- IX $<$ 1 or IX+M-1 $>$ M_ (= DESCX(1)) (assuming X(IX:IX+M-1,$*$) is to be operated on),
- JX $<$ 1 or JX+N-1 $>$ N_ (= DESCX(2)), (assuming X($*$,JX:JX+N-1) is to be operated on),
- MB_ (=DESCX(3)) $<$ 1 or NB_ (=DESCX(4)) $<$ 1,
- RSRC_ (=DESCX(5)) $<$ 0 or RSRC_ $\geq$ P (number of process rows),
- CSRC_ (=DESCX(6)) $<$ 0 or CSRC_ $\geq$ Q (number of process columns),
- LLD_ (=DESCX(8)) $<$ the local number of rows in the array pointed to by X,
- INCX $\neq$ 1 and INCX $\neq$ M_ (= DESCX(1)) (Only for vector operands),
- CTXT_X (=DESCX(7)) $\neq$ CTXT_Y (=DESCY(7)) with X and Y distributed matrix operands.

If a routine is called with an invalid value for any of its arguments, then it must report the fact and terminate the execution of the program. In the model implementation, each routine, on detecting an error, calls a common error-handling routine PBERROR(), passing to it the current BLACS context, the name of the routine and the number of the first argument that is in error. If an error is detected in the j-th entry of a descriptor array, which is the i-th argument in the parameter list, the number passed to PBERROR() has been arbitrarily chosen to be $100*i+j$. This allows the user to distinguish an error on a descriptor entry from an error on a scalar argument. For efficiency purposes, the PBLAS routines only perform a local validity check of their argument list. If an error is detected in at least one process of the current context, the program execution is stopped.

A global validity check of the input arguments passed to a PBLAS routine must be performed in the higher-level calling procedure. To demonstrate the need and cost of global checking, as well as the reason why this type of checking is not performed in the PBLAS, consider the following example: the value of a global input argument is legal but differs from one process to another. The results are unpredictable. In order to detect this kind of error situation, a synchronization point would be necessary, which may result in a significant performance degradation. Since every process must call the same routine to perform the desired operation successfully, it is natural and safe to restrict somewhat the amount of checking operations performed in the PBLAS routines.

Specialized implementations may call system-specific exception-handling facilities, either via an auxiliary routine PBERROR or directly from the routine. In addition, the testing programs can take advantage of this exception-handling mechanism by simulating specific erroneous input argument lists and then verifying that particular errors are correctly detected.

4 Specifications of the PBLAS

4.1 Argument Declarations

Type, dimension and description for variables occurring in the subroutine specifications are as follows:

```
CHARACTER*1       SIDE, UPLO, TRANS, TRANSA, TRANSB, DIAG
INTEGER           IA, IB, IC, INCX, INCY, INDX, IX, IY
INTEGER           JA, JB, JC, JX, JY, M, N, K
INTEGER           DESCA( 8 ), DESCB( 8 ), DESCC( 8 )
INTEGER           DESCX( 8 ), DESCY( 8 )
```

For routines whose second letter is a S:

```
REAL              ALPHA, AMAX, ASUM, BETA, DOT, NRM2
REAL              A( * ), B( * ), C( * ), X( * ), Y ( * )
```

For routines whose second letter is a D:

```
DOUBLE PRECISION   ALPHA, AMAX, ASUM, BETA, DOT, NRM2
DOUBLE PRECISION   A( * ), B( * ), C( * ), X( * ), Y ( * )
```

For routines whose second letter is a C:

```
REAL               AMAX, ASUM, DOTC, DOTU, NRM2
COMPLEX            ALPHA, BETA
COMPLEX            A( * ), B( * ), C( * ), X( * ), Y ( * )
```

except for PCHER and PCHERK where the scalars α and β are real so that the first declaration above is replaced by:

```
REAL               ALPHA, BETA
```

and for PCHER2K α is complex and β is real, so this is replaced by:

```
COMPLEX            ALPHA
REAL               BETA
```

For routines whose second letter is a Z:

```
DOUBLE PRECISION   AMAX, ASUM, DOTC, DOTU, NRM2
COMPLEX*16         ALPHA, BETA
COMPLEX*16         A( * ), B( * ), C( * ), X( * ), Y( * )
```

or equivalently,

```
DOUBLE PRECISION   AMAX, ASUM, DOTC, DOTU, NRM2
DOUBLE COMPLEX     ALPHA, BETA
DOUBLE COMPLEX     A( * ), B( * ), C( * ), X( * ), Y( * )
```

except for PZHER and PZHERK where the scalars α and β are real so that the first declaration above is replaced by:

```
DOUBLE PRECISION   ALPHA, BETA
```

and for PCHER2K where α is complex and β is real, so this is replaced by:

```
COMPLEX*16         ALPHA
DOUBLE PRECISION   BETA
```

or equivalently,

```
DOUBLE COMPLEX     ALPHA
DOUBLE PRECISION   BETA
```

4.2 Vector-Vector Operations

In the following sections, no distinction is made between a column or a row of a matrix. Both are denoted by the word "vector". We define $vec_N(X)$ by

$$vec_N(X) = \begin{cases} \texttt{X(IX, JX : JX + N - 1)} \text{ if } \texttt{INCX} = \texttt{DESCX(1)} \text{ and} \\ \texttt{X(IX : IX + N - 1, JX)} \text{ if } \texttt{INCX} = \texttt{1}. \end{cases}$$

1. Vector swap:

   ```
   P□SWAP( N, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY )
   ```

   Operation:

   $$vec_N(X) \leftrightarrow vec_N(Y)$$

2. Vector scaling:

   ```
   P□SCAL( N, ALPHA, X, IX, JX, DESCX, INCX )
   ```

   Operation:

   $$vec_N(X) \leftarrow \alpha \; vec_N(X)$$

3. Vector copy:

   ```
   P□COPY( N, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY )
   ```

   Operation:

   $$vec_N(Y) \leftarrow vec_N(X)$$

4. Vector addition:

   ```
   P□AXPY( N, ALPHA, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY )
   ```

   Operation:

   $$vec_N(Y) \leftarrow \alpha \; vec_N(X) + vec_N(Y)$$

5. Dot products:

   ```
   P□DOT□( N, DOT, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY )
   ```

   Operation: for the PSDOT, PDDOT, PCDOTU or PZDOTU routines,

$$\texttt{DOT} \leftarrow vec_N(X)^T vec_N(Y)$$

For the PCDOTC or PZDOTC routines,

$$\texttt{DOT} \leftarrow vec_N(X)^H vec_N(Y)$$

6. Vector 2-norm:

```
P□NRM2( N, NORM2, X, IX, JX, DESCX, INCX )
```

Operation:

$$\texttt{NORM2} \leftarrow \|vec_N(X)\|_2$$

7. Sum of absolute value of vector entries:

```
P□ASUM( N, ASUM, X, IX, JX, DESCX, INCX )
```

Operation:

$$\texttt{ASUM} \leftarrow \|re(vec_N(X))\|_1 + \|im(vec_N(X))\|_1$$

8. Index and value of vector entry having maximum absolute value:

```
P□AMAX( N, AMAX, INDX, X, IX, JX, DESCX, INCX )
```

Operation:

$$\left\{\begin{array}{ll} \texttt{INDX} \leftarrow & 1^{st}\ k \ni |re(vec_N(X)_k)| + |im(vec_N(X)_k)| \\ & = max(|re(vec_N(X)_i)| + |im(vec_N(X)_i)|) \\ \texttt{AMAX} \leftarrow & vec_N(X)_k \end{array}\right.$$

4.3 Matrix-Vector Operations

In the following sections, $sub_{M,N}(A)$ denotes the submatrix A(IA:IA+M-1,JA:JA+N-1).

1. General matrix-vector products:

```
P□GEMV( TRANS, M, N, ALPHA, A, IA, JA, DESCA, X, IX, JX, DESCX, INCX,
        BETA, Y, IY, JY, DESCY, INCY )
```

Operation:

$$\text{TRANS = 'N'} \qquad vec_M(Y) \leftarrow \alpha\ sub_{M,N}(A) vec_N(X) + \beta\ vec_M(Y)$$

$$\text{TRANS = 'T'} \qquad vec_N(Y) \leftarrow \alpha\ sub_{M,N}(A)^T vec_M(X) + \beta\ vec_N(Y)$$

$$\text{TRANS = 'C'} \qquad vec_N(Y) \leftarrow \alpha\ sub_{M,N}(A)^H vec_M(X) + \beta\ vec_N(Y)$$

2. Matrix-vector products where the matrix is real or complex symmetric or complex Hermitian:

```
P□SYMV( UPLO, N, ALPHA, A, IA, JA, DESCA, X, IX, JX, DESCX, INCX,
        BETA, Y, IY, JY, DESCY, INCY )
P□HEMV( UPLO, N, ALPHA, A, IA, JA, DESCA, X, IX, JX, DESCX, INCX,
        BETA, Y, IY, JY, DESCY, INCY )
```

   Operation: $sub_{N,N}(A)$ is symmetric for the `P□SYMV` routines and Hermitian for the `P□HEMV` routines:

$$vec_N(Y) \leftarrow \alpha\ sub_{N,N}(A) vec_N(X) + \beta\ vec_N(Y)$$

3. Triangular matrix-vector products:

```
P□TRMV( UPLO, TRANS, DIAG, N, A, IA, JA, DESCA, X, IX, JX, DESCX, INCX )
```

   Operation: $sub_{N,N}(A)$ denotes a triangular submatrix:

$$\text{TRANS = 'N'} \qquad vec_N(X) \leftarrow sub_{N,N}(A) vec_N(X)$$

$$\text{TRANS = 'T'} \qquad vec_N(X) \leftarrow sub_{N,N}(A)^T vec_N(X)$$

$$\text{TRANS = 'C'} \qquad vec_N(X) \leftarrow sub_{N,N}(A)^H vec_N(X)$$

4. Solution of triangular system of equations:

```
P□TRSV( UPLO, TRANS, DIAG, N, A, IA, JA, DESCA, X, IX, JX, DESCX, INCX )
```

   Operation: $sub_{N,N}(A)$ denotes a triangular submatrix:

$$\text{TRANS = 'N'} \qquad vec_N(Y) \leftarrow sub_{N,N}(A)^{-1} sub_N(X)$$

$$\text{TRANS = 'T'} \qquad vec_N(Y) \leftarrow sub_{N,N}(A)^{-T} sub_N(X)$$

$$\text{TRANS = 'C'} \qquad vec_N(Y) \leftarrow sub_{N,N}(A)^{-H} sub_N(X)$$

5. Rank-1 updates of a general matrix:

```
P□GER□(  M, N, ALPHA, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY,
         A, IA, JA, DESCA )
```

Operation: for the PSGER, PDGER, PCGERU or PZGERU routines,

$$sub_{M,N}(A) \leftarrow \alpha\ vec_M(X) vec_N(Y)^T + sub_{M,N}(A)$$

For the PCGERC or PZGERC routines,

$$sub_{M,N}(A) \leftarrow \alpha\ vec_M(X) vec_N(Y)^H + sub_{M,N}(A)$$

6. Rank-1 updates of a real or complex symmetric or complex Hermitian matrix:

```
P□SYR( UPLO, N, ALPHA, X, IX, JX, DESCX, INCX, A, IA, JA, DESCA )
P□HER( UPLO, N, ALPHA, X, IX, JX, DESCX, INCX, A, IA, JA, DESCA )
```

Operation: for the P□SYR routines, $sub_{N,N}(A)$ is symmetric:

$$sub_{N,N}(A) \leftarrow \alpha\ vec_N(X) vec_N(X)^T + sub_{N,N}(A)$$

For the P□HER routines, $sub_{N,N}(A)$ is Hermitian,

$$sub_{N,N}(A) \leftarrow \alpha\ vec_N(X) vec_N(X)^H + sub_{N,N}(A)$$

7. Rank-2 updates of a real or complex symmetric or complex Hermitian matrix:

```
P□SYR2(  UPLO, N, ALPHA, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY,
         A, IA, JA, DESCA )
P□HER2(  UPLO, N, ALPHA, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY,
         A, IA, JA, DESCA )
```

Operation: for the P□SYR2 routines, $sub_{N,N}(A)$ is symmetric,

$$sub_{N,N}(A) \leftarrow \alpha\ vec_N(X) vec_N(Y)^T + \alpha\ vec_N(Y) vec_N(X)^T + sub_{N,N}(A)$$

For the P□HER2 routines, $sub_{N,N}(A)$ is Hermitian,

$$sub_{N,N}(A) \leftarrow \alpha\ vec_N(X) vec_N(Y)^H + \overline{\alpha}\ vec_N(Y) vec_N(X)^H + sub_{N,N}(A)$$

4.4 Matrix-Matrix Operations

1. General matrix-matrix products:

```
P□GEMM( TRANSA, TRANSB, M, N, K, ALPHA, A, IA, JA, DESCA,
        B, IB, JB, DESCB, BETA, C, IC, JC, DESCC )
```

Operation: in the following table $sub(C)$ denotes $sub_{M,N}(C)$, $sub(A)$ denotes $sub_{M,K}(A)$ when `TRANSA='N'` and $sub_{K,M}(A)$ otherwise, finally $sub(B)$ denotes $sub_{K,N}(B)$ when `TRANSB='N'` and $sub_{N,K}(B)$ otherwise.

	`TRANSA = 'N'`	`TRANSA = 'T'`	`TRANSA = 'C'`
`TRANSB = 'N'`	$sub(C) \leftarrow \beta\ sub(C)$ $+\ \alpha\ sub(A)sub(B)$	$sub(C) \leftarrow \beta\ sub(C)$ $+\ \alpha\ sub(A)^T sub(B)$	$sub(C) \leftarrow \beta\ sub(C)$ $+\ \alpha\ sub(A)^H sub(B)$
`TRANSB = 'T'`	$sub(C) \leftarrow \beta\ sub(C)$ $+\ \alpha\ sub(A)sub(B)^T$	$sub(C) \leftarrow \beta\ sub(C)$ $+\ \alpha\ sub(A)^T sub(B)^T$	$sub(C) \leftarrow \beta\ sub(C)$ $+\ \alpha\ sub(A)^H sub(B)^T$
`TRANSB = 'C'`	$sub(C) \leftarrow \beta\ sub(C)$ $+\ \alpha\ sub(A)sub(B)^H$	$sub(C) \leftarrow \beta\ sub(C)$ $+\ \alpha\ sub(A)^T sub(B)^H$	$sub(C) \leftarrow \beta\ sub(C)$ $+\ \alpha\ sub(A)^H sub(B)^H$

(In the real case the values `'T'` and `'C'` have the same meaning).

2. Matrix-matrix products where one matrix is real or complex symmetric or complex Hermitian:

```
P□SYMM( SIDE, UPLO, M, N, ALPHA, A, IA, JA, DESCA,
        B, IB, JB, DESCB, BETA, C, IC, JC, DESCC )
P□HEMM( SIDE, UPLO, M, N, ALPHA, A, IA, JA, DESCA,
        B, IB, JB, DESCB, BETA, C, IC, JC, DESCC )
```

Operation: $sub_{M,M}(A)$ when `SIDE='L'` and $sub_{N,N}(A)$ when `SIDE='R'` is symmetric for the `P□SYMM` routines, Hermitian for the `P□HEMM` routines:

`SIDE = 'L'` $\quad sub_{M,N}(C) \leftarrow \alpha\ sub_{M,M}(A)sub_{M,N}(B) + \beta\ sub_{M,N}(C)$

`SIDE = 'R'` $\quad sub_{M,N}(C) \leftarrow \alpha\ sub_{M,N}(B)sub_{N,N}(A) + \beta\ sub_{M,N}(C)$

3. Rank-k updates of a real or complex symmetric or complex Hermitian matrix:

```
P□SYRK( UPLO, TRANS, N, K, ALPHA, A, IA, JA, DESCA, BETA, C, IC, JC, DESCC )
P□HERK( UPLO, TRANS, N, K, ALPHA, A, IA, JA, DESCA, BETA, C, IC, JC, DESCC )
```

Operation: for the `P□SYRK` routines, $sub_{N,N}(C)$ is symmetric,

TRANS = 'N'	$sub_{N,N}(C) \leftarrow \alpha\ sub_{N,K}(A) sub_{N,K}(A)^T + \beta\ sub_{N,N}(C)$
TRANS = 'T'	$sub_{N,N}(C) \leftarrow \alpha\ sub_{K,N}(A)^T sub_{K,N}(A) + \beta\ sub_{N,N}(C)$

For the P□HERK routines, $sub_{N,N}(C)$ is Hermitian,

TRANS = 'N'	$sub_{N,N}(C) \leftarrow \alpha\ sub_{N,K}(A) sub_{N,K}(A)^H + \beta\ sub_{N,N}(C)$
TRANS = 'C'	$sub_{N,N}(C) \leftarrow \alpha\ sub_{K,N}(A)^H sub_{K,N}(A) + \beta\ sub_{N,N}(C)$

(In the real cases the values 'T' and 'C' have the same meaning. In the complex case TRANS='C' is not allowed in P□SYRK, and TRANS='T' is not allowed in P□HERK).

4. Rank-$2k$ updates of a real or complex symmetric or complex Hermitian matrix:

```
P□SYR2K( UPLO, TRANS, N, K, ALPHA, A, IA, JA, DESCA,
         B, IB, JB, DESCB, BETA, C, IC, JC, DESCC )
P□HER2K( UPLO, TRANS, N, K, ALPHA, A, IA, JA, DESCA,
         B, IB, JB, DESCB, BETA, C, IC, JC, DESCC )
```

Operation: for the P□SYR2K routines, $sub_{N,N}(C)$ is symmetric,

TRANS = 'N'	$sub_{N,N}(C) \leftarrow \alpha\ sub_{N,K}(A) sub_{N,K}(B)^T + \alpha\ sub_{N,K}(B) sub_{N,K}(A)^T + \beta\ sub_{N,N}(C)$
TRANS = 'T'	$sub_{N,N}(C) \leftarrow \alpha\ sub_{K,N}(A)^T sub_{K,N}(B) + \alpha\ sub_{K,N}(B)^T sub_{K,N}(A) + \beta\ sub_{N,N}(C)$

For the P□HER2K routines, $sub(C)$ is Hermitian,

TRANS = 'N'	$sub_{N,N}(C) \leftarrow \alpha\ sub_{N,K}(A) sub_{N,K}(B)^H + \overline{\alpha}\ sub_{N,K}(B) sub_{N,K}(A)^H + \beta\ sub_{N,N}(C)$
TRANS = 'C'	$sub_{N,N}(C) \leftarrow \alpha\ sub_{K,N}(A)^H sub_{K,N}(B) + \overline{\alpha}\ sub_{K,N}(B)^H sub_{K,N}(A) + \beta\ sub_{N,N}(C)$

(In the real cases the values 'T' and 'C' have the same meaning. In the complex case TRANS='C' is not allowed in P□SYR2K, and TRANS='T' is not allowed in P□HER2K).

5. Matrix transposition

```
P□TRAN□( M, N, ALPHA, A, IA, JA, DESCA, BETA, C, IC, JC, DESCC )
```

Operation: for the PSTRAN, PDTRAN, PCTRANU or PZTRANU routines,

$$sub_{M,N}(C) \leftarrow \beta\ sub_{M,N}(C) + \alpha\ sub_{N,M}(A)^T$$

For the PCTRANC or PZTRANC routines,

$$sub_{M,N}(C) \leftarrow \beta\ sub_{M,N}(C) + \alpha\ sub_{N,M}(A)^H$$

6. Triangular matrix-matrix products:

```
P□TRMM(  SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, IA, JA, DESCA,
         B, IB, JB, DESCB )
```

Operation: in the following table, $sub(B)$ denotes $sub_{M,N}(B)$, $sub(A)$ denotes the $sub_{M,M}(A)$ when SIDE='L' and $sub_{N,N}(A)$ when SIDE='R'. $sub(A)$ is triangular:

	SIDE = 'L'	SIDE = 'R'
TRANSA = 'N'	$sub(B) \leftarrow \alpha\ sub(A)sub(B)$	$sub(B) \leftarrow \alpha\ sub(B)sub(A)$
TRANSA = 'T'	$sub(B) \leftarrow \alpha\ sub(A)^T sub(B)$	$sub(B) \leftarrow \alpha\ sub(B)sub(A)^T$
TRANSA = 'C'	$sub(B) \leftarrow \alpha\ sub(A)^H sub(B)$	$sub(B) \leftarrow \alpha\ sub(B)sub(A)^H$

(In the real case the values 'T' and 'C' have the same meaning.)

7. Solution of triangular systems of equations:

```
P□TRSM(  SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, IA, JA, DESCA,
         B, IB, JB, DESCB )
```

Operation: in the following table, $sub(B)$ denotes $sub_{M,N}(B)$, $sub(A)$ denotes the $sub_{M,M}(A)$ when SIDE='L' and $sub_{N,N}(A)$ when SIDE='R'. $sub(A)$ is triangular:

	SIDE = 'L'	SIDE = 'R'
TRANSA = 'N'	$sub(B) \leftarrow \alpha\ sub(A)^{-1} sub(B)$	$sub(B) \leftarrow \alpha\ sub(B)sub(A)^{-1}$
TRANSA = 'T'	$sub(B) \leftarrow \alpha\ sub(A)^{-T} sub(B)$	$sub(B) \leftarrow \alpha\ sub(B)sub(A)^{-T}$
TRANSA = 'C'	$sub(B) \leftarrow \alpha\ sub(A)^{-H} sub(B)$	$sub(B) \leftarrow \alpha\ sub(B)sub(A)^{-H}$

(In the real case the values 'T' and 'C' have the same meaning.)

5 Implementation

To support and encourage the use of the PBLAS, we describe here two software components of this package which emphasize the software quality aspects used during the development phase, as well as explain our reasons to believe in the reliability and robustness of these routines:

1. A model implementation of the subprograms has been written in ANSI C [25], mainly for its dynamic memory allocation management features, and FORTRAN 77. The FORTRAN 77 BLAS enable the PBLAS to be used on any machine for which the BLACS are available.

2. Testing and timing programs have been designed to ensure that implementations conform to the specifications and have been correctly installed.

5.1 The Model Implementation

While most of the local computations are performed by the BLAS and the communication is handled by the BLACS, the PBLAS is in fact only responsible for organizing the distributed computations. A typical PBLAS subroutine locally checks the coherency and the validity of its input arguments, translates these global parameters into their local equivalents and performs the basic operations using an optimally shaped adaptive procedure. Note that most of the PBLAS routines currently assume the data to be aligned. Various routines have different alignment restrictions. For instance, some routines will require that two matrices start at the same process row or column, while others may require only the block size to be the same. In the next version of the PBLAS, some of these restrictions have been removed and the remaining restrictions will be evaluated by user feedback.

5.1.1 Efficiency.

At the lowest level, the efficiency of the PBLAS is determined by the local performance of the BLAS and the BLACS. In addition, depending on the shape of its input and output distributed matrix operands, the PBLAS select the best algorithm in terms of data transfer across the process grid. Transparent to the user, this relatively simple selection process ensures high efficiency independent from the actual computation performed.

For example, there are algorithms [10, 19, 22], for matrix-matrix products like PUMMA which are much more efficient for equally sized input/output matrices. Some of these algorithms require a very large amount of workspace making them impractical for library purposes. However, a simple implementation of common matrix multiplication operations has recently been proven to be highly efficient and scalable [26]. These algorithms, called SUMMA, have the advantage of requiring much less workspace than PUMMA. These algorithms have, in some sense, already been implemented in terms of internal routines to the PBLAS [9]. Therefore, this work [26] will allow us to improve and generalize the model implementation. However, when one of the matrix operands is "thin" or "fat", the current model implementation employs different algorithms which are more efficient in the overall number of messages exchanged on the network, and are also usually much more economical in terms of workspace.

The current model implementation of the Level 3 PBLAS decides which algorithm to use depending on the shape of the matrix operands. This decision, however, could also be based on the amount of memory available during the execution, the local BLAS performance, and machine constants such as the latency and bandwidth of the network [4].

Internally, the PBLAS currently rely on routines requiring certain alignment properties to be satisfied [9]. These properties have been chosen so that maximum efficiency can be obtained on these restricted operations. Consequently, when redistribution or re-alignment of input or output data has to be performed some performance will be lost. So far, the PBLAS do not perform such redistribution or alignment of the input/output matrix or vector operands when necessary. However, the PBLAS routines would provide greater flexibility and would be more similar in functionality to the BLAS if these operations where provided. The question of making the PBLAS more flexible remains open and its answer largely depends on the needs of the user community.

5.1.2 Auxiliary Subprograms.

It is well known [4, 13, 26] that certain algorithms based on a two-dimensional block-cyclic data distribution scheme become more efficient and scalable when appropriate communication topologies are used for the broadcast and global combine operations [4, 13, 26]. For example, pipelining the broadcast operation along the rows of the process grid improves the efficiency and scalability of the LU factorization algorithm [4, 13]. The BLACS topologies allow the user to optimize communication patterns for these particular operations. A default topology can also be selected. The list of BLACS topologies as well as the different possible scopes are documented in [14]. In order to set this low level information, the PBLAS provide two routines having the following FORTRAN 77 interface:

```
SUBROUTINE PTOPSET( ICTXT, OP, SCOPE, TOP )
SUBROUTINE PTOPGET( ICTXT, OP, SCOPE, TOP )

INTEGER            ICTXT
CHARACTER*1        OP, SCOPE, TOP
```

PTOPSET assigns the BLACS topology [14] TOP to be used in the communication operations OP along the scope specified by SCOPE. PTOPGET returns the BLACS topology TOP used in the communication operations OP along the scope specified by SCOPE. Application examples of these routines are given in appendix B. The BLACS provide broadcast (OP='B') and global combine (OP='C') operations to which different topologies are associated. The scope refers to the group of processes involved in such a BLACS operation. It indicates whether a process row (SCOPE='R'), process column (SCOPE='C'), or the entire grid (SCOPE='A') will participate in these operations.

In addition, the PBLAS provide a subroutine to dispose of the PBLAS buffer allocated in every process's dynamic memory. Its FORTRAN 77 interface is:

```
SUBROUTINE PBFREEBUF()
```

5.2 Testing

Master test programs have been designed, developed and included with the submitted code. This package consists of several main programs and a set of subprograms generating test data and comparing the results with data obtained by element-wise computations or the sequential BLAS. These testing programs assume the correctness of the BLAS and the BLACS routines; it is therefore highly recommended to run the testing programs provided with both of these packages before performing any PBLAS test. A separate test program exists for each of the four data types (*real, complex, double precision and complex∗16*) as well as each PBLAS level. All test programs conform to the same pattern with only the minimum necessary changes. These programs have been designed not merely to check whether the model implementation has been correctly installed, but also to serve as a validation tool and a modest debugging aid for any specialized implementation.

These programs have the following features:

- the parameters of the test problems and the names of the subprogram to be tested are specified by means of an input data file, which can easily be modified for debugging,
- the data for the test problems are generated internally and the results are checked internally,
- the programs check that no arguments are changed by the routines except the designated output scalar, vector or matrix. All input error exits (caused by illegal parameter values) are tested,
- the programs generate a concise summary report of the tests as well as pertinent error messages when needed.

Input data files are supplied with every test program, but installers and implementors must be alert to the possible need to extend or modify them. Values of the elements of the matrix operands are uniformly distributed over $(-1.0, 1.0)$. Care is taken to ensure that the data have full working accuracy. Elements in the distributed matrices that are not to be referenced by a subprogram are either checked after exiting the routine or set to a "rogue" value (-10.0^{10}) to increase the likelihood that a reference to them will be detected. If a computational error is reported and an element of the computed result is of order 10.0^{10}, then the routine has almost certainly referenced the wrong element of the array.

After each call to a subprogram being tested, its operation is checked in two ways. First, each of its input arguments, including all elements of the distributed operands, is checked to see if it has been altered by the subprogram. If any argument, other than the specified elements of the result scalar, vector or matrix, has been modified, an error is reported. This check includes the supposedly unreferenced elements of the distributed matrices. Second, the resulting scalar, vector or matrix computed by the subprogram is compared with the corresponding result obtained by the sequential BLAS or by simple Fortran code. We do not expect exact agreement because the two results are not necessarily computed by the same sequences of floating point operations. We do, however, expect the differences to be small relative to working precision. The error bounds are then the same as the ones used in the BLAS testers. A more detailed description of those tests can be found in [11, 12]. The test ratio is determined by scaling these error bounds by the inverse of machine epsilon ϵ^{-1}. This ratio is compared with a constant threshold value defined in the input data file. Test ratios greater than the threshold are flagged as "suspect". On the basis of the BLAS experience a threshold value of 16 is recommended. The precise value is not critical. Errors in the routines are most likely to be errors in array indexing, which will almost certainly lead to a totally wrong result. A more subtle potential error is the use of a single precision variable in a double precision computation. This is likely to lead to a loss of half the machine epsilon. The test programs regard a test ratio greater than $\epsilon^{-\frac{1}{2}}$ as an error.

The PBLAS testing programs are thus very similar to what has been done for the BLAS. However, it was necessary to slightly depart from the way the BLAS testing programs operate due to the difficulties inherent to the testing of programs written for distributed-memory computers.

The first obstacle is due to the significant increase of testing parameters. Indeed, programs for distributed-memory computers need to be tested for virtually any number of processes. Moreover, it should also be possible to vary the data distribution parameters such as the block sizes defined in Sect. 3.2. These facts motivated the decision to permit a user configurable set of tests for every routine. Consequently, one can test the PBLAS with any possible machine configuration as well as data layout.

The second more subtle difficulty is due to the routines producing an output scalar such as P□NRM2. Because of the block-cyclic decomposition properties and the fact that vector operands are so far restricted to a matrix row or column, it follows that only one process row or column will own the input vector. This process row or column is subsequently called the vector scope by analogy with the BLACS terminology. The question becomes: which processes should get the correct result ? It experimentally appeared convenient to broadcast the result to every process in the vector scope only and set it to zero elsewhere. If this scalar is needed by every process in the grid, it is the user's responsibility to broadcast it. Consequently, such routines need only to be called by the processes in the vector scope. Moreover, this appropriate specification to what is needed by the ScaLAPACK routines introduces a slight ambiguity when one wants to compute for example the norm of a column of a 1-by-N distributed matrix. Indeed, this 1-column can equivalently be seen as a row subsection containing one entry. In practice, this case rarely occurs. Should it happen, the PBLAS routines return the correct result only in the process owning the input vector operand and zero in every other grid process.

Finally, there are special challenges associated with writing and testing numerical software to be executed on networks containing heterogeneous processors [4], i.e., processors which perform floating point arithmetic differently. This includes not just machines with different floating point formats and semantics such as Cray computers and workstations performing IEEE standard floating point arithmetic, but even supposedly identical machines running different compilers or even just different compiler options. Moreover, on such networks, floating point data transfers between two processes may require a data conversion phase and thus a possible loss of accuracy. It is therefore impractical, error-prone and difficult to compare supposedly identical computed scalars on such heterogeneous networks. As a consequence, the validity and correctness of the tests performed can only be guaranteed for networks of processors with identical floating point formats.

6 Rationale

In the design of all levels of the PBLAS, as with the BLAS, one of the main concerns is to keep both the calling sequences and the range of options limited, while at the same time maintaining sufficient functionality. This clearly implies a compromise, and a good judgement is vital if the PBLAS are to be accepted as a useful standard. In this section we discuss some of the reasoning behind the decisions we have made.

A large amount of sequential linear algebra software relies on the BLAS. Because software reusability is one of our major concerns, we wanted the BLAS and PBLAS interfaces to be as similar as possible. Consequently, only one routine, the matrix transposition, has been added to the PBLAS, since this operation is much more complex to perform in a distributed-memory environment [8].

One of the major differences between the BLAS and the PBLAS is likely to be found in the Level 1 routines. Indeed, the functions of the former have been replaced by subroutines in the latter. In our model implementation, the top-level routines are written in C, thus it was not possible to return a scalar anywhere else than in the argument list and at the same time to have the routines callable by C or FORTRAN programs. Moreover, it is useful for the `P□AMAX` routines to return not only the value of the element of maximum absolute value but also its global index. This contradicts the principle that a function only returns a single value, thus the function became a subroutine.

The scalar values returned by the Level 1 PBLAS routines `P□DOT□`, `P□NRM2`, `P□ASUM` and `P□AMAX` are only correct in the scope of their operands and zero elsewhere. For example, when INCX is equal to one, only the column of processes having part of the vector operands gets the correct results. This decision was made for efficiency purposes. It is, however, very easy to have this information broadcast across the process mesh by directly calling the appropriate BLACS routine. Consequently, these particular routines do not need to be called by any other processes other than the ones in the scope of their operands. With this exception in mind, the PBLAS follow an SPMD programming model and need to be called by every process in the current BLACS context to work correctly.

Nevertheless, there are a few more exceptions in the current model implementation, where some computations local to a process row or column can be performed by the PBLAS, without having every process in the grid calling the routine. For example, the rank-1 update performed in the LU factorization presented in the next section, involves data which is contained by only one process column. In this case, to maintain the efficiency of the factorization it is important to have this particular operation performed only within one process column. In other words, when a PBLAS routine is called by every process in the grid, it is required that the code operates successfully as specified by the SPMD programming model. However, it is also necessary that the PBLAS routines recognize the scope of their operands in order to save useless communication and synchronization costs when possible. This specific part of the PBLAS specifications remains an open question.

A few features supported by the PBLAS underlying tools [9] have been intentionally hidden. For instance, a block of identical vectors operands are sometimes replicated across process rows or columns. When such a situation occurs, it is possible to save some communication and computation operations. The PBLAS interface could provide such operations, for example, by setting the origin process coordinate in the array descriptor to -1 (see Sect. 3.2). Such features, for example, would be useful in the ScaLAPACK routines responsible for applying a block of Householder vectors to a matrix. Indeed, these Householder vectors need to be broadcast to every process row or column before being applied. Whether or not this feature should be supported by the PBLAS is still an open question.

We have adhered to the conventions of the BLAS in allowing an increment argument to be associated with each distributed vector so that a vector could, for example, be a row of a matrix. However, negative increments or any value other than 1 or $DESC\_(1)$ are not supported by our current model implementation. The negative increments -1 and $-DESC\_(1)$ should be relatively easy to support. It is still unclear how it would be possible to take advantage of this added complexity and if other increment values should be supported.

The presence of BLACS contexts associated with every distributed matrix provides the ability to have separate "universes" of message passing. The use of separate communication contexts by distinct libraries (or distinct library invocations) such as the PBLAS insulates communication internal to the library execution from external communication. When more than one descriptor array is present in the argument list of a routine in the PBLAS, it is required that the BLACS context entries must be equal (see Sect. 3.3). In other words, the PBLAS do not perform "intra-context" operations.

We have not included specialized routines to take advantage of packed storage schemes for symmetric, Hermitian, or triangular matrices, nor of compact storage schemes for banded matrices. As with the BLAS no check has been included for singularity, or near singularity, in the routines for solving triangular systems of equations. The requirements for such a test depend on the application and so we felt that this should not be included, but should instead be performed outside the triangular solve.

For obvious software reusability reasons we have tried to adhere to the conventions of, and maintain consistency with, the sequential BLAS. However, we have deliberately departed from this approach by explicitly passing the global indices and using *array descriptors*. Indeed, it is our experience that using a "local indexing" scheme for the interface makes the use of these routines much more complex from the user's point of view. Our implementation of the PBLAS emphasizes the mathematical view of a matrix over its storage. In fact, other block distributions may be able to reuse both the interface and the descriptor described in this paper without change. Fundamentally different distributions may require modifications of the descriptor, but the interface should remain unchanged.

The model implementation in its current state provides sufficient functionality for the use of the PBLAS modules in the ScaLAPACK library. However, as we mentioned earlier in this paper, there are still a few details that remain open questions and may easily be accommodated as soon as more experience with these codes is reported. Hopefully, the comments and suggestions of the user community will help us to make these last decisions so that this proposal can be made more rigorous and adequate to the user's needs.

Finally, it is our experience that porting sequential code built on the top of the BLAS to distributed memory machines using the PBLAS is much simpler than writing the parallel code from scratch (see Sect. 7). Taking the BLAS proposals as our model for software design was in our opinion a way to ensure the same level of software quality for the PBLAS.

7 Applications and Use of the PBLAS

The PBLAS is a component of the ScaLAPACK library. As such, installing the PBLAS library is part of the ScaLAPACK installation procedure. This process is described in [5]. However, the PBLAS will likely become a stand-alone package in the near future, similar to what has been done for the BLAS and LAPACK libraries [2]. In which case, the installation procedure of the stand-alone PBLAS library will be very close to what has been done for the ScaLAPACK library. This section contains code fragments that demonstrate what needs to be done in order to call a PBLAS routine. Then, we show how the PBLAS routines can be used in order to write a parallel linear system solver for distributed memory MIMD computers. This code is in fact a slightly simplified version of the ScaLAPACK code.

7.1 Use of the PBLAS

In order to call a PBLAS routine, it is necessary to initialize the BLACS and create the process grid. This can be done by calling the routine `BLACS_GRIDINIT` (see [14] for more details). The following segment of code will arrange four processes into a 2×2 process grid. When running on platforms such as PVM [20], where the number of computational nodes available is unknown a priori, it is necessary to call the routine `BLACS_SETUP`, so that copies (3 in our example) of the main program can be spawned on the virtual machine. Finally, in order to ensure a safe coexistence with other parallel libraries using a distinct message passing layer, such as MPI [17], the BLACS routine `BLACS_GET` queries for an eventual system context (see [14] for more details).

```
      INTEGER            IAM, ICTXT, NPROCS
*
*     (...)
*
      CALL BLACS_PINFO( IAM, NPROCS )
*
      IF( NPROCS.LT.1 ) THEN
         NPROCS = 4
         CALL BLACS_SETUP( IAM, NPROCS )
      END IF
*
      CALL BLACS_GET( -1, 0, ICTXT )
      CALL BLACS_GRIDINIT( ICTXT, 'Row-major', 2, 2 )
*
*     (...)
*
```

Moreover, to convey the data distribution information to the PBLAS, the descriptor of the matrix operands should be set. The ScaLAPACK library contains a tool routine called `DESCINIT` for that purpose. This routine takes as arguments the 8-integer (descriptor) array to be initialized, as well as the 8 entries to be used. Finally, an error flag is set on output to detect if an incoherent descriptor entry is passed to this routine. `DESCINIT` should be called by every process in the grid.

We present in the following code fragment the descriptor initialization phase as well as a call to a PBLAS routine. This sample program performs the matrix multiplication:

$$\mathtt{C}(1:4,1:4) \leftarrow \mathtt{A}(1:4,1:4) * \mathtt{B}(1:4,1:4).$$

This example program is to be run on four processes arranged in a 2×2 process grid. The matrices A, B and C are 5×5 matrices partitioned into 2×2 blocks. We choose the process of coordinates $(0,0)$ to be the owner of the first entries of the matrices A, B and C. The mapping of these matrices is identical to the example of Fig. 1 given in Sect. 3.2.

```
      INTEGER            INFO, NMAX, LDA, LDB, LDC, NMAX
      PARAMETER          ( NMAX = 3, LDA = NMAX, LDB = NMAX, LDC = NMAX )
*
      INTEGER            DESCA( 8 ), DESCB( 8 ), DESCC( 8 )
      DOUBLE PRECISION   A( NMAX, NMAX ), B( NMAX, NMAX ), C( NMAX, NMAX )
*
*     (...)
*
*     Initialize the array descriptors for the matrices A, B and C
*
      CALL DESCINIT( DESCA, 5, 5, 2, 2, 0, 0, ICTXT, LDA, INFO )
      CALL DESCINIT( DESCB, 5, 5, 2, 2, 0, 0, ICTXT, LDB, INFO )
      CALL DESCINIT( DESCC, 5, 5, 2, 2, 0, 0, ICTXT, LDC, INFO )
*
*     (...)
*
      CALL PDGEMM( 'No transpose', 'No transpose', 4, 4, 4, 1.0D+0,
     $             A, 1, 1, DESCA, B, 1, 1, DESCB, 0.0D+0,
     $             C, 1, 1, DESCC )
*
*     (...)
*
```

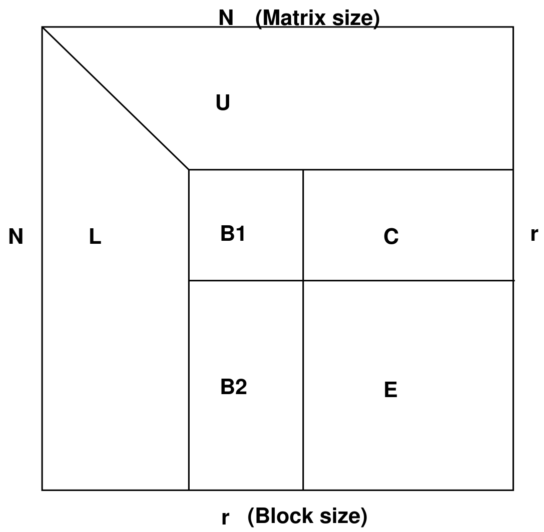
Finally, it is recommended to release the resources allocated by the BLACS and the PBLAS just before ending the program segment using the BLACS and the PBLAS. Note that the routine `BLACS_GRIDEXIT` will free the resources associated with a particular context, while the routine `BLACS_EXIT` will free all BLACS resources (see [14] for more details).

```
      CALL PBFREEBUF()
*
      CALL BLACS_GRIDEXIT( ICTXT )
*
      CALL BLACS_EXIT( 0 )
```

7.2 Solving Linear Systems via LU Factorization

The primary applications of the PBLAS are in implementing algorithms of numerical linear algebra in terms of operations on submatrices (or blocks). Therefore, provisions have been made to easily port sequential programs built on top of the BLAS onto distributed memory computers. Note that the ScaLAPACK library provides a set of tool routines, which the user might find useful for this purpose.

In the following diagram we illustrate how the PBLAS routines can be used to port a simple algorithm of numerical linear algebra, namely solving systems of linear equations via LU factorization. Note that more examples may be found in the ScaLAPACK library.



To obtain a parallel implementation of the LU factorization of a N-by-N matrix, we started with a variant of the right-looking LAPACK LU factorization routine given in appendix B.1. This algorithm proceeds along the diagonal of the matrix by first factorizing a block B of r columns at a time, with pivoting if necessary. Then a triangular solve and a rank-r update are performed on the rest of the matrix. This process continues recursively with the updated matrix.

For $k = 1$ *to* N/r do

Factor panel B with pivoting, (P□AMAX, P□SWAP, P□GER□)

Apply pivots to the remainder of the matrix, (P□SWAP)

Solve $C := B1^{-1}C$, (Triangular solve, P□TRSM)

Update $E := E - B2 * C$, (Rank-r update, P□GEMM)

End for;

From the application programmer's point of view, it is conceptually simple to translate the serial version of the code into its parallel equivalent. Translating BLAS calls to PBLAS calls primarily consists of the following steps: a `'P'` has to be inserted in front of the routine name, the leading dimensions should be replaced by the global *array descriptors*, and the global indices into the distributed matrices should be inserted as separate parameters in the calling sequence:

```
      CALL DGEMM( 'No transpose', 'No transpose', M-J-JB+1, N-J-JB+1,
     $            JB, -ONE, A( J+JB, J ), LDA, A( J, J+JB ), LDA, ONE,
     $            A( J+JB, J+JB ), LDA )
```

$$\downarrow$$

```
      CALL PDGEMM( 'No transpose', 'No transpose', M-J-JB+JA, N-J-JB+JA,
     $             JB, -ONE, A, I+JB, J, DESCA, A, I, J+JB, DESCA, ONE,
     $             A, I+JB, J+JB, DESCA )
```

This simple translation process considerably simplifies the implementing phase of linear algebra codes built on top of the BLAS. Moreover, the global view of the matrix operands allows the user to be concerned only with the numerical details of the algorithms and a minimum number of important details necessary to programs written for distributed-memory computers.

The resulting parallel code is given in appendix B along with the serial code. These codes are very similar as most of the details of the parallel implementation such as communication and synchronization have been hidden at lower levels of the software.

In addition, the underlying block-cyclic decomposition scheme ensures good load-balance, and thus performance and scalability. In the particular example of the LU factorization, it is possible to take advantage of other parallel algorithmic techniques such as pipelining and the overlapping of computation and communication operations. Because the factorization and pivoting phases of the algorithm described above are much less computational intensive than its update phase, it is intuitively suitable to communicate the pivot indices as soon as possible to all processes in the grid, especially to those who possess the next block of columns to be factorized. In this way the update phase can be started as early as possible [13]. Such a pipelining effect can easily be achieved within PBLAS based codes by using ring topologies along process rows for the broadcast operations. These particular algorithmic techniques are enabled by the PBLAS auxiliary routines `PTOPGET` and `PTOPSET` (see Sect. 5.1.2). They notably improve the performance and the scalability of the parallel implementation.

References

[1] M. Aboelaze, N. Chrisochoides, and E. Houstis. "The parallelization of Level 2 and 3 BLAS Operations on Distributed Memory Machines". Technical Report CSD-TR-91-007, Purdue University, West Lafayette, IN, 1991.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *"LAPACK Users' Guide, Second Edition"*. SIAM, Philadelphia, PA, 1995.

[3] R. Brent and P. Strazdins. "Implementation of BLAS Level 3 and LINPACK Benchmark on the AP1000". *Fujitsu Scientific and Technical Journal*, 5(1):61–70, 1993.

[4] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. "ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance". Technical Report UT CS-95-283, LAPACK Working Note #95, University of Tennessee, 1995.

[5] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. "Installation Guide for ScaLAPACK". Technical Report UT CS-95-280, LAPACK Working Note #93, University of Tennessee, 1995.

[6] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. C. Whaley. "The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines". Technical Report UT CS-94-246, LAPACK Working Note #80, University of Tennessee, 1994.

[7] J. Choi, J. Dongarra, R. Pozo, and D. Walker. "ScaLAPACK: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers". In *Proceedings of Fourth Symposium on the Frontiers of Massively Parallel Computation (McLean, Virginia)*, pages 120–127. IEEE Computer Society Press, Los Alamitos, California, 1992. (also LAPACK Working Note #55).

[8] J. Choi, J. Dongarra, and D. Walker. Parallel matrix transpose algorithms on distributed memory concurrent computers. In *Proceedings of Fourth Symposium on the Frontiers of Massively Parallel Computation (McLean, Virginia)*, pages 245–252. IEEE Computer Society Press, Los Alamitos, California, 1993. (also LAPACK Working Note #65).

[9] J. Choi, J. Dongarra, and D. Walker. "PB-BLAS: A Set of Parallel Block Basic Linear Algebra Subroutines". In *"Proceedings of the Scalable High Performance Computing Conference"*, pages 534–541, Knoxville, TN, 1994. IEEE Computer Society Press.

[10] J. Choi, J. Dongarra, and D. Walker. "PUMMA: Parallel Universal Matrix Multiplication Algorithms on Distributed Memory Concurrent Computers". *Concurrency: Practice and Experience*, 6(7):543–570, 1994. (also LAPACK Working Note #57).

[11] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. "A Set of Level 3 Basic Linear Algebra Subprograms". *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[12] J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. "Algorithm 656: An extended Set of Basic Linear Algebra Subprograms: Model Implementation and Test Programs". *ACM Transactions on Mathematical Software*, 14(1):18–32, 1988.

[13] J. Dongarra, R. van de Geijn, and D. Walker. "Scalability Issues in the Design of a Library for Dense Linear Algebra". *Journal of Parallel and Distributed Computing*, 22(3):523–537, 1994. (also LAPACK Working Note #43).

[14] J. Dongarra and R. C. Whaley. "A User's Guide to the BLACS v1.0". Technical Report UT CS-95-281, LAPACK Working Note #94, University of Tennessee, 1995.

[15] A. Elster. "Basic Matrix Subprograms for Distributed Memory Systems". In D. Walker and Q. Stout, editors, *"Proceedings of the Fifth Distributed Memory Computing Conference"*, pages 311–316. IEEE Press, 1990.

[16] R. Falgout, A. Skjellum, S. Smith, and C. Still. "The Multicomputer Toolbox Approach to Concurrent BLAS". *submitted to Concurrency: Practice and Experience*, 1993. (preprint).

[17] Message Passing Interface Forum. "MPI: A Message Passing Interface Standard". *International Journal of Supercomputer Applications and High Performance Computing*, 8(3–4), 1994.

[18] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker. *"Solving Problems on Concurrent Processors"*, volume 1. Prentice Hall, Englewood Cliffs, N.J, 1988.

[19] G. Fox, S. Otto, and A. Hey. "Matrix Algorithms on a Hypercube I: Matrix Multiplication". *Parallel Computing*, 3:17–31, 1987.

[20] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *"PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing"*. The MIT Press, Cambridge, Massachusetts, 1994.

[21] R. Hanson, F. Krogh, and C. Lawson. "A Proposal for Standard Linear Algebra Subprograms". *ACM SIGNUM Newsl.*, 8(16), 1973.

[22] S. Huss-Lederman, E. Jacobson, A. Tsao, and G. Zhang. "Matrix Multiplication on the Intel Touchstone DELTA". *Concurrency: Practice and Experience*, 6(7):571–594, 1994.

[23] C. Koebel, D. Loveman, R. Schreiber, G. Steele, and M. Zosel. *"The High Performance Fortran Handbook"*. The MIT Press, Cambridge, Massachusetts, 1994.

[24] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. "Basic Linear Algebra Subprograms for Fortran Usage". *ACM Transactions on Mathematical Software*, 5(3):308–323, 1979.

[25] H. Schildt. *"The Annoted ANSI C standard. American National Standard for Programming Languages – C. ANSI/ISO 9899-1990"*. OsBorne, Berkeley, CA, 1990.

[26] R. van de Geijn and J. Watts. "SUMMA: Scalable Universal Matrix Multiplication Algorithm". Technical Report UT CS-95-286, LAPACK Working Note #96, University of Tennessee, 1995.

[27] R. C. Whaley. "Basic Linear Algebra Communication Subprograms: Analysis and Implementation Across Multiple Parallel Architectures". Technical Report UT CS-94-234, LAPACK Working Note #73, University of Tennessee, 1994.

A Questions for the Community

For convenience we summarize here those questions on which we would particularly welcome feedback:

- Should the alignment restrictions in the current implementation be removed, or do the PBLAS provide sufficient functionality the way they are ? (see Sect. 5.1)
- When a PBLAS routine is called by every process in the grid, it is required that the code operates successfully accordingly to the SPMD programming model. However, it is also necessary that the PBLAS routines recognize the scope of their operands for efficiency purposes. Is it reasonable to slightly depart from the SPMD programming model for efficiency purposes ? (see Sect. 6)
- Should the PBLAS be able to recognize and take advantage of replicated operands across process rows or columns ? For example, a column replicated vector is a vector distributed over the rows of a process column, and every other column owns an aligned copy of that vector. If such an operand is to be set on output, should all the distinct copies of the replicated array be updated ? (see Sect. 6)
- Should other vector increment values (e.g `INCX`) be supported beside 1 and `DESCX(1)` ? (see Sect. 6)
- Is the current set of PBLAS routines sufficient or should we consider adding more routines and increase the PBLAS functionality and usefulness ? (see Sect. 6)

B Code Examples

B.1 Sequential LU Factorization

```
      SUBROUTINE DGETRF( M, N, A, LDA, IPIV, INFO )
*
*  LU factorization of a M-by-N matrix A using partial pivoting with
*  row interchanges.
*
      INTEGER            INFO, LDA, M, N, IPIV( * )
      DOUBLE PRECISION   A( LDA, * )
*
      INTEGER            I, IINFO, J, JB, NB
      PARAMETER          ( NB = 64 )
      EXTERNAL           DGEMM, DGETF2, DLASWP, DTRSM
      INTRINSIC          MIN
*
      DO 20 J = 1, MIN(M,N), NB
         JB = MIN( MIN(M,N)-J+1, NB )
*
*        Factor diagonal block and test for exact singularity.
*
         CALL DGETF2( M-J+1, JB, A(J,J), LDA, IPIV(J), IINFO )
*
*        Adjust INFO and the pivot indices.
*
         IF( INFO.EQ.0 .AND. IINFO.GT.0 ) INFO = IINFO + J - 1
         DO 10 I = J, MIN(M,J+JB-1)
            IPIV(I) = J - 1 + IPIV(I)
   10    CONTINUE
*
*        Apply interchanges to columns 1:J-1 and J+JB:N.
*
         CALL DLASWP( J-1, A, LDA, J, J+JB-1, IPIV, 1 )
         IF( J+JB.LE.N ) THEN
            CALL DLASWP( N-J-JB+1, A(1,J+JB), LDA, J, J+JB-1, IPIV, 1 )
*
*           Compute block row of U and update trailing submatrix.
*
            CALL DTRSM( 'Left', 'Lower', 'No transpose', 'Unit', JB,
     $                  N-J-JB+1, 1.0D+0, A(J,J), LDA, A(J,J+JB), LDA )
            IF( J+JB.LE.M )
     $         CALL DGEMM( 'No transpose', 'No transpose', M-J-JB+1,
     $                     N-J-JB+1, JB, -1.0D+0, A(J+JB,J), LDA,
     $                     A(J,J+JB), LDA, 1.0D+0, A(J+JB,J+JB), LDA )
         END IF
   20 CONTINUE
      RETURN
*
      END
```

B.2 Parallel LU Factorization

```
      SUBROUTINE PDGETRF( M, N, A, IA, JA, DESCA, IPIV, INFO )
*
      INTEGER            IA, INFO, JA, M, N, DESCA( 8 ), IPIV( * )
      DOUBLE PRECISION   A( * )
*
*  LU factorization of a M-by-N distributed matrix A(IA:IA+M-1,JA:JA+N-1)
*  using partial pivoting with row interchanges.
*
      INTEGER            I, IINFO, J, JB
      EXTERNAL           IGAMN2D, PTOPSET, PDGEMM, PDGETF2, PDLASWP, PDTRSM
      INTRINSIC          MIN
*
      CALL PTOPSET( 'Broadcast', 'Row', 'S-ring' )
      DO 10 J = JA, JA+MIN(M,N)-1, DESCA( 4 )
         JB = MIN( MIN(M,N)-J+JA, DESCA( 4 ) )
         I = IA + J - JA
*
*        Factor diagonal block and test for exact singularity.
*
         CALL PDGETF2( M-J+JA, JB, A, I, J, DESCA, IPIV, IINFO )
         IF( INFO.EQ.0 .AND. IINFO.GT.0 ) INFO = IINFO + J - JA
*
*        Apply interchanges to columns JA:J-JA and J+JB:JA+N-1.
*
         CALL PDLASWP( 'Forward', 'Rows', J-JA, A, IA, JA, DESCA,
     $                 I, I+JB-1, IPIV )
         IF( J-JA+JB+1.LE.N ) THEN
            CALL PDLASWP( 'Forward', 'Rows', N-J-JB+JA, A, IA, J+JB,
     $                    DESCA, I, I+JB-1, IPIV )
*
*           Compute block row of U and update trailing submatrix.
*
            CALL PDTRSM( 'Left', 'Lower', 'No transpose', 'Unit', JB,
     $                   N-J-JB+JA, 1.0D+0, A, I, J, DESCA, A, I, J+JB,
     $                   DESCA )
            IF( J-JA+JB+1.LE.M ) THEN
     $         CALL PDGEMM( 'No transpose', 'No transpose', M-J-JB+JA,
     $                      N-J-JB+JA, JB, -1.0D+0, A, I+JB, J, DESCA, A,
     $                      I, J+JB, DESCA, 1.0D+0, A, I+JB, J+JB, DESCA )
         END IF
   10 CONTINUE
      IF( INFO.EQ.0 ) INFO = MIN(M,N) + 1
      CALL IGAMN2D( ICTXT, 'Row', ' ', 1, 1, INFO, 1, I, J, -1, -1, MYCOL )
      IF( INFO.EQ.MIN(M,N)+1 ) INFO = 0
      CALL PTOPSET( 'Broadcast', 'Row', ' ' )
*
      RETURN
*
      END
```

B.3 Parallel General Linear System Solve

```
      SUBROUTINE PDGETRS( TRANS, N, NRHS, A, IA, JA, DESCA, IPIV, B,
     $                    IB, JB, DESCB )
*
      CHARACTER          TRANS
      INTEGER            IA, IB, IDUM1, JA, JB, N, NRHS
      INTEGER            DESCA( * ), DESCB( * ), DESCIP( 8 ), IPIV( * )
      DOUBLE PRECISION   A( * ), B( * )
*
      LOGICAL            LSAME
      INTEGER            NUMROC
      EXTERNAL           DESCSET, LSAME, NUMROC, PDLAPIV, PDTRSM
*
      IF( N.EQ.0 .OR. NRHS.EQ.0 ) RETURN
      CALL DESCSET( DESCIP, DESCA( 1 ) + DESCA( 3 )*NPROW, 1, DESCA( 3 ),
     $      1, DESCA( 5 ), MYCOL, ICTXT, DESCA( 3 ) +
     $      NUMROC( DESCA( 1 ), DESCA( 3 ), MYROW, DESCA( 5 ), NPROW ) )
*
      IF( LSAME( TRANS, 'N' ) ) THEN
*
*        Solve A * X = B. Apply row interchanges to the right hand sides.
*        Solve L*X = B, overwriting B with X.
*        Solve U*X = B, overwriting B with X.
*
         CALL PDLAPIV( 'Forward', 'Row', 'Col', N, NRHS, B, IB, JB,
     $                 DESCB, IPIV, IA, 1, DESCIP, IDUM1 )
         CALL PDTRSM( 'Left', 'Lower', 'No transpose', 'Unit', N, NRHS,
     $                1.0D+0, A, IA, JA, DESCA, B, IB, JB, DESCB )
         CALL PDTRSM( 'Left', 'Upper', 'No transpose', 'Non-unit', N,
     $                NRHS, 1.0D+0, A, IA, JA, DESCA, B, IB, JB, DESCB )
      ELSE
*
*        Solve A' * X = B.  Solve U'*X = B, overwriting B with X.
*        Solve L'*X = B, overwriting B with X.
*        Apply row interchanges to the solution vectors.
*
         CALL PDTRSM( 'Left', 'Upper', 'Transpose', 'Non-unit', N, NRHS,
     $                1.0D+0, A, IA, JA, DESCA, B, IB, JB, DESCB )
         CALL PDTRSM( 'Left', 'Lower', 'Transpose', 'Unit', N, NRHS,
     $                1.0D+0, A, IA, JA, DESCA, B, IB, JB, DESCB )
         CALL PDLAPIV( 'Backward', 'Row', 'Col', N, NRHS, B, IB, JB,
     $                 DESCB, IPIV, IA, 1, DESCIP, IDUM1 )
      END IF
*
      RETURN
*
      END
```

C Quick Reference to the PBLAS

Level 1 PBLAS

	dim scalar / vector / vector		prefixes
P□SWAP	(N, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY)	$x \leftrightarrow y$	S, D, C, Z
P□SCAL	(N, ALPHA, X, IX, JX, DESCX, INCX)	$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
P□COPY	(N, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY)	$y \leftarrow x$	S, D, C, Z
P□AXPY	(N, ALPHA, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY)	$y \leftarrow \alpha x + y$	S, D, C, Z
P□DOT	(N, DOT, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY)	$dot \leftarrow x^T y$	S, D
P□DOTU	(N, DOTU, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY)	$dotu \leftarrow x^T y$	C, Z
P□DOTC	(N, DOTC, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY)	$dotc \leftarrow x^H y$	C, Z
P□NRM2	(N, NORM2, X, IX, JX, DESCX, INCX)	$norm2 \leftarrow \|x\|_2$	S, D, SC, DZ
P□ASUM	(N, ASUM, X, IX, JX, DESCX, INCX)	$asum \leftarrow \|re(x)\|_1 + \|im(x)\|_1$	S, D, SC, DZ
P□AMAX	(N, AMAX, INDX, X, IX, JX, DESCX, INCX)	$indx \leftarrow 1^{st}\ k \ni \|Re(x_k)\| + \|Im(x_k)\| = max(\|Re(x_i)\| + \|Im(x_i)\|) = amax$	S, D, C, Z

Level 2 PBLAS

	options / dim / scalar / matrix / vector / scalar / vector		
P□GEMV	(TRANS, M, N, ALPHA, A, IA, JA, DESCA, X, IX, JX, DESCX, INCX, BETA, Y, IY, JY, DESCY, INCY)	$y \leftarrow \alpha op(A)x + \beta y, op(A) = A, A^T, A^H, A - m \times n$	S, D, C, Z
P□HEMV	(UPLO, N, ALPHA, A, IA, JA, DESCA, X, IX, JX, DESCX, INCX, BETA, Y, IY, JY, DESCY, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
P□SYMV	(UPLO, N, ALPHA, A, IA, JA, DESCA, X, IX, JX, DESCX, INCX, BETA, Y, IY, JY, DESCY, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
P□TRMV	(UPLO, TRANS, DIAG, N, A, IA, JA, DESCA, X, IX, JX, DESCX, INCX)	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x,$	S, D, C, Z
P□TRSV	(UPLO, TRANS, DIAG, N, A, IA, JA, DESCA, X, IX, JX, DESCX, INCX)	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x,$	S, D, C, Z

	options / dim / scalar / vector / vector / matrix		
P□GER	(M, N, ALPHA, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY, A, IA, JA, DESCA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	S, D
P□GERU	(M, N, ALPHA, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY, A, IA, JA, DESCA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	C, Z
P□GERC	(M, N, ALPHA, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY, A, IA, JA, DESCA)	$A \leftarrow \alpha xy^H + A, A - m \times n$	C, Z
P□HER	(UPLO, N, ALPHA, X, IX, JX, DESCX, INCX, A, IA, JA, DESCA)	$A \leftarrow \alpha xx^H + A$	C, Z
P□HER2	(UPLO, N, ALPHA, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY, A, IA, JA, DESCA)	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
P□SYR	(UPLO, N, ALPHA, X, IX, JX, DESCX, INCX, A, IA, JA, DESCA)	$A \leftarrow \alpha xx^T + A$	S, D
P□SYR2	(UPLO, N, ALPHA, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY, A, IA, JA, DESCA)	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D

Level 3 PBLAS

	options / dim / scalar / matrix / matrix / scalar / matrix		
P□GEMM	(TRANSA, TRANSB, M, N, K, ALPHA, A, IA, JA, DESCA, B, IB, JB, DESCB, BETA, C, IC, JC, DESCC)	$C \leftarrow \alpha op(A)op(B) + \beta C, op(X) = X, X^T, X^H, C - m \times n$	S, D, C, Z
P□SYMM	(SIDE, UPLO, M, N, ALPHA, A, IA, JA, DESCA, B, IB, JB, DESCB, BETA, C, IC, JC, DESCC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^T$	S, D, C, Z
P□HEMM	(SIDE, UPLO, M, N, ALPHA, A, IA, JA, DESCA, B, IB, JB, DESCB, BETA, C, IC, JC, DESCC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^H$	C, Z
P□SYRK	(UPLO, TRANS, N, K, ALPHA, A, IA, JA, DESCA, BETA, C, IC, JC, DESCC)	$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C, C - n \times n$	S, D, C, Z
P□HERK	(UPLO, TRANS, N, K, ALPHA, A, IA, JA, DESCA, BETA, C, IC, JC, DESCC)	$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C, C - n \times n$	C, Z
P□SYR2K	(UPLO, TRANS, N, K, ALPHA, A, IA, JA, DESCA, B, IB, JB, DESCB, BETA, C, IC, JC, DESCC)	$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C, C \leftarrow \alpha A^T B + \alpha B^T A + \beta C, C - n \times n$	S, D, C, Z
P□HER2K	(UPLO, TRANS, N, K, ALPHA, A, IA, JA, DESCA, B, IB, JB, DESCB, BETA, C, IC, JC, DESCC)	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C, C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C, C - n \times n$	C, Z
P□TRAN	(M, N, ALPHA, A, IA, JA, DESCA, BETA, C, IC, JC, DESCC)	$C \leftarrow \beta C + \alpha A^T, A - n \times m, C - m \times n$	S, D
P□TRANU	(M, N, ALPHA, A, IA, JA, DESCA, BETA, C, IC, JC, DESCC)	$C \leftarrow \beta C + \alpha A^T, A - n \times m, C - m \times n$	C, Z
P□TRANC	(M, N, ALPHA, A, IA, JA, DESCA, BETA, C, IC, JC, DESCC)	$C \leftarrow \beta C + \alpha A^H, A - n \times m, C - m \times n$	C, Z
P□TRMM	(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, IA, JA, DESCA, B, IB, JB, DESCB)	$B \leftarrow \alpha op(A)B, B \leftarrow \alpha B op(A), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z
P□TRSM	(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, IA, JA, DESCA, B, IB, JB, DESCB)	$B \leftarrow \alpha op(A^{-1})B, B \leftarrow \alpha B op(A^{-1}), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z

Meaning of prefixes

S - `REAL`	C - `COMPLEX`
D - `DOUBLE PRECISION`	Z - `COMPLEX*16` (may not be supported by all machines)

Level 2 and Level 3 PBLAS Matrix Types

GE - GEneral
SY - SYmmetric
HE - HErmitian
TR - TRiangular

Level 2 and Level 3 PBLAS Options

Dummy options arguments are declared as CHARACTER*1 and may be passed as character strings.
TRANS□ = 'No transpose', 'Transpose', 'Conjugate transpose', (X, X^T, X^H)
UPLO = 'Upper triangular', 'Lower Triangular'
DIAG = 'Non-unit triangular', 'Unit triangular'
SIDE = 'Left' or 'Right' (A or op(A) on the left, or A or op(A) on the right)

For real matrices, TRANS□ = 'T' and TRANS□ = 'C' have the same meaning.

For Hermitian matrices, TRANS□='T' is not allowed.

For complex symmetric matrices, TRANS□='C' is not allowed.

Obtaining the software via netlib

In order to get instructions for downloading the PBLAS, send email to `netlib@ornl.gov` and in the body of the message type `send index from scalapack`.

Send comments, questions to `scalapack@cs.utk.edu`.

Array Descriptor, Increment

The array descriptor *DESCA* is an integer array of dimension 8. It describes the two-dimensional block-cyclic mapping of the matrix A.

The two first entries are the dimensions of the matrix (row, column). The third and fourth entries are the row- and column block sizes used to distribute the matrix. The fifth and the sixth are the coordinates of the process containing the first entry of the matrix. The seventh entry is the BLACS context in which the computation takes place. The last entry contains the leading dimension of the local array containing the matrix elements.

The increment specified for vectors is always global. So far only 1 and DESCA(1) are supported.

References

J. Dongarra and R. C. Whaley, LAPACK, Working Note 94, *A User's Guide to the BLACS v1.0*, Computer Science Dept. Technical Report CS-95-281, University of Tennessee, Knoxville, March, 1995. To receive a postscript copy, send email to netlib@ornl.gov and in the mail message type: `send lawn94.ps from lapack/lawns`.

J. Choi, J. Dongarra, and D. Walker, *PB-BLAS: A Set of Parallel Block Basic Linear Algebra Subroutines*, Proceedings of Scalable High Performance Computing Conference (Knoxville, TN), pp. 534-541, IEEE Computer Society Press, May 23-25, 1994.

J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker and R. C. Whaley, LAPACK, Working Note 95, *ScaLAPACK: A Scalable Linear Algebra library for Distributed Memory Concurrent Computers - Design Issues and Performance*, Computer Science Dept. Technical Report CS-95-283, University of Tennessee, Knoxville, March 1995. To receive a postscript copy, send email to netlib@ornl.gov and in the mail message type: `send lawn95.ps from lapack/lawns`.

J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker and R. C. Whaley, LAPACK, Working Note 100, *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms*, Computer Science Dept. Technical Report CS-95-292, University of Tennessee, Knoxville, July 1995. To receive a postscript copy, send email to netlib@ornl.gov and in the mail message type: `send lawn100.ps from lapack/lawns`.

Parallel

Basic

Linear

Algebra

Subprograms

Release 1.0

University of Tennessee

March 28, 1995

A Quick Reference Guide