Toward a Proposal for a set of Parallel Basic Linear Algebra Subprograms

BLAS Technical Forum*

February 25, 1997

Abstract

This (draft) document aims at identifying a number of issues to be discussed in order to formulate a proposal for a set of Parallel Basic Linear Algebra Subprograms. This collection of subprograms is targeted at elementary distributed dense and sparse linear algebra operations. These subprograms can in turn be used to develop parallel libraries for a large variety of distributed memory concurrent computers.

*Edited by Antoine Petitet

Contents

1 Introduction **3**

2 Notation **3**

2.1 Operands 3

2.2 Elementary Operations 4

2.2.1 Addition 4

2.2.2 Multiplication 4

2.2.3 Indexes 4

3 Parallel Basic Linear Algebra Operations **4**

4 Data Distributions **5**

5 Inter-Intra Context Operations **5**

6 Parallel Basic Linear Algebra Algorithms **6**

7 Alignment Restriction - Redistribution **6**

8 Replication – Persistent Objects **6**

9 Divide and Conquer – Task Scheduling **7**

10 I/O – Data Generation **7**

11 The Dense Case: The ScaLAPACK Experience **7**

12 Rationale **8**

1 Introduction

This (draft) document aims at identifying a number of issues to be discussed in order to formulate a comprehensive or limited proposal for a set of Parallel Basic Linear Algebra Subprograms. The ultimate goal of this effort is to provide specifications of distributed kernels suitable for their implementation as well as their use in higher level software components. *Functionality, efficiency and flexibility* are the three essential concepts influencing this proposal. Functionality and efficiency are motivated by obvious usefulness reasons. Flexibility is often considered as less important, even though, it is likely to be the most essential criterion for the acceptance of this proposal by a large user community. It is our hope that this proposal will initiate discussions among the computer science community so that this project will best reflect its needs.

2 Notation

2.1 Operands

In this document, six main operands are considered:

- Indexes (i, j),
- Scalars (α, β and γ),
- Multi-indexes (I, J),
- Multi-scalars (, and Δ),
- Vectors (x, y and z),
- Multi-vectors (X, Y and Z),
- Matrices (A, B, C and T).

All these objects can be regarded as arrays of numbers. Indexes or scalars are just one by one arrays. Multi-indexes, multi-scalars and vectors are one dimensional arrays. Multi-vectors and matrices are two-dimensional arrays of numbers. A multi-vector can be regarded as either a collection of column vectors or a collection of row vectors. Vectors, multi-vectors, matrices may be dense or sparse. Matrices can in addition be diagonal, symmetric, hermitian triangular, orthogonal, banded or any meaningful combinations of those.

The above notation identify mathematical objects without assuming any particular kind of data distribution. It is considered that the distribution of these different objects is inherently part of them and does not impact the operations described in this proposal, but only their eventual implementation. Indexes, multi-indexes multi-scalars and multi-vectors mainly arise in the sparse context.

2.2 Elementary Operations

This section aims at defining the elementary addition and multiplication operations that can be performed between the different operands defined above; that is, for example, the meaning of adding a scalar to a multiscalar.

2.2.1 Addition

Adding two operands of the same type, say two scalars, produces an operand of the same type. Similarly, one can define say the addition of a scalar to a multi-scalar by first expanding the scalar to a multi-scalar, and then adding two operands of the same type.

2.2.2 Multiplication

Multiplication is harder to define, but it can be done in a similar fashion as presented for the addition. For instance, consider the multiplication of a multi-scalar by a vector. In this case, the vector should be first expanded to a row or column multi-vector. It is then easy to multiply a multi-scalar by a multi-vector by using multiple instances of the usual scalar by a vector multiplication. Other instances of multiplication can similarly be defined. It is important to note that when such an expansion occur the result might be of a different type. For example, multiplying a vector by a multi-scalar produces a multi-vector. Consequently, specific care must be taken in the specification and creation of such output operands. In addition, one may as well declare some of the operations involving different types of operands as undefined. For example, it is not obvious what kind of operand should be produced by the multiplication of a multi-scalar by a matrix, if not a "multi-matrix".

2.2.3 Indexes

Indexes and multi-indexes are "passive" operands in the sense that they usually do not interact on each other. Consequently, the addition or multiplication of indexes or multi-indexes are not defined.

3 Parallel Basic Linear Algebra Operations

The parallel basic linear algebra operations have equivalents in single and double precision arithmetic. In addition, the operands may be real or complex. However, it is also useful to consider some mixed real and complex operations such as scaling a complex vector by a real scalar. Finally, in the complex case one should also consider operating on a transposed operand as opposed to a conjugate transposed operand. Similarly, it is also worth considering to operate on the conjugated operand.

Operations involving multivectors have vector equivalents. Multivectors can be operated on with scalars, multiscalars, multivectors or matrices. It is relatively straightforward to produce a list of operations comparable to the BLAS ones or their extension. Operations involving matrices and multi-vectors are of use to the dense and sparse contexts. Matrix-matrix operations are almost uniquely of interest in a dense setting.

In a parallel setting, it is meaningful to consider performing multiple operations on a single data. For example, one may be willing to generate and apply different plane rotations on a unique matrix. These operations are relatively difficult to specify. However, they should be considered.

4 Data Distributions

A large number of data distribution schemes have been devised and studied in the context of parallel linear algebra operations. It is probable that more schemes may be invented or defined in the future. This is particularly true when considering the current machine architecture evolution. Obviously, this changing state of affair does not simplify the task of this committee. Consequently, it is wise to confine the data distribution details to the operands themselves as opposed to the operations. Still, the data distribution schemes that may be retained for some implementation must be parametrized to allow for efficiency on machines featuring highly different communication and computation rates. Whether a parallel basic linear algebra operation should require all its operands to be distributed according to the same scheme is unclear.

5 Inter-Intra Context Operations

Each operand is to be distributed on a subset of processes. This subset of processes constitutes a communication space where the communication operations are performed. The specification of an operation involving at least two operands must then specify if the operands belong to the same communication space (intra-context operations) or if the operands may belong to distinct communication spaces (inter-context operations).

Inter-context operations are more general, and thus more flexible. They also require a synchronization phase between the communication contexts, so they are slightly less efficient than intra context operations. In both kinds of operations, to a different extent though, data re-alignment is an issue to be considered for obvious flexibility reasons.

Inter- and intra-context operations require each operand to be bound to at least one communication context. It does not seem reasonable to support operands that could belong simultaneously to multiple communication contexts.

6 Parallel Basic Linear Algebra Algorithms

A very large variety of algorithms are available to perform parallel basic linear algebra operations. A good example of this is given by the matrix-matrix multiplication. Similarly as in the serial case, one may consider to provide different variants for efficiency reasons. Memory usage is usually a limiting factor, that may justify the selection of a particular algorithm. Indeed, memory greedy algorithms affect their ease-of-use. For example, Strassen's method among others are worth considering for its computational savings. This applies indirectly to other block-operations that may use internally such a matrix-matrix multiply. This section can be generalized to other operations such as "fast triangular solves (fan-in)".

7 Alignment Restriction - Redistribution

Alignment restrictions significantly affect the ease-of-use of a parallel library. First, the documentation needs to reflect these complicated conditions. Second, from a user point of view it considerably limit the use of essential kernels. This aspect of things should be taken into account within the specifications of each kernel. Obviously this has an impact on the interface of an eventual model implementation. It is recommended to specify whenever possible operations that do not require any alignment conditions to be satisfied.

8 Replication – Persistent Objects

Replication is a shy attempt to support persistent objects. Replication means that an operation should be able to take advantage of the duplication of certain operands in certain process subsets. This information may be carried out by the operand itself. Another way of proceeding is to consider a set of specific operations for replicated operations.

Replicated scalars, vectors or multi-vectors often occur. They are usually tight to another operand. A good example is the pivot vector in the LU factorization. Depending on your data distribution and/or algorithms, you may find useful to replicate this pivot vector in each process column. Obviously, this is not mandatory, but saves communication operations during the solve phase. This last example brings up another issue. Since this pivot vector is inherently tied to the factorized matrix, it would be useful to be able to express such a relationship between two different data objects.

Replication seems an interesting feature to consider especially in an homogeneous setting. Indeed, it allows for redundant computations as well as communication savings. In an heterogeneous environment, maintaining the coherency of such replicated operands is a challenge that should be addressed.

Persistent objects are ultimately more powerful than data replication. However, the software machinery to support them is also more complicated. Still, they should be considered as possible operands.

9 Divide and Conquer – Task Scheduling

In dense or sparse applications, it is often the case that multiple problems arise in processes that are already busy when other are idle. Tools for creating, scheduling and distributing tasks across processes are necessary for a number of parallel algorithms. Such a functionality seems suitable to be introduced at the PBLAS level for its own use as well as higher level software components.

10 I/O – Data Generation

From a user perspective, it is important to provide elementary tools to generate or retrieve from disk the distributed data to be operated on. Portability is a critical issue for I/O operations. Such tools are also responsible for distributing the data if necessary.

11 The Dense Case: The ScaLAPACK Experience

In its current form, the dense part of the ScaLAPACK library contains a set of Parallel Basic Linear Algebra Subprograms for dense kernels. These subprograms provide a comparable functionality to the sequential BLAS equivalent. Their interface has been designed to simplify and enable the re-use of the sequential LAPACK software library.

These subprograms operate on operands distributed in a block-cyclic manner. Block cyclic distribution is beneficial because of its scalability load balance and communication properties. The block-partitioned computation of higher level components then proceeds in consecutive order just like a conventional serial algorithm. This essential property of the block cyclic data layout explains why the ScaLAPACK design has been able to reuse the numerical and software expertise of the sequential LAPACK library.

A descriptor is associated with each operand. It is assumed that all operands are distributed on the same process grid. In other words, only intra-context operations are supported. This descriptor is a simple array of integers describing the block-cyclic layout of the distributed matrices. Each operand, i.e., a subvector or a submatrix, is globally specified by its global size and and starting indexes as well as its descriptor. This operand specification is very similar to the array notation and allow for extensive code reusability. This is illustrated by the following conceptual example. The call to the sequential routine DGEXXX is easily translated into its parallel equivalent. DESCA is the descriptor associated to the "distributed matrix" A, one for good at the beginning of the user's program.

```
CALL  DGEXXX( M, N, A( IA, JA ), LDA )
                        ↓
CALL PDGEXXX( M, N, A, IA, JA, DESCA )
```

The first entry of the descriptor identifies its type. This feature allows one to provide another set of PBLAS routines for a different data distribution without modifying higher level software

components. This interface uses a very simple data structure, namely an integer array. This simplicity allow this software to be easily used by programs written in other programming languages such as C++ or HPF.

The algorithms currently implemented in the current version of the matrix-matrix operations are shaped-adaptive procedures. This means that depending on the operands' shapes, different algorithms are selected depending on their respective efficiency. This set of dense parallel basic linear algebra subprograms has been shown to deliver high performance as well as an efficient tool for the scalable implementation of classic matrix factorizations and reductions. In addition, these same kernels have been shown to be useful for the implementation of the out-of-core linear solvers currently available in the ScaLAPACK software library.

12 Rationale

This present document aims at identifying issues that impact the specification of parallel dense and sparse basic linear algebra subprograms. It is in its current form largely incomplete. The large number of distributed data objects that are of interest in a parallel setting magnify the specification task of this committee. By limiting the number of possible operands, and distribution schemes, it has been shown feasible by many research groups to provide sufficient functionality for a restricted application domain. This is particularly true for the dense case or even some sparse applications, even if the software volume becomes rapidly large and thus non trivial to maintain. This practical consideration should influence the work of this committee.

In its current form, this document aims at clearly identifying the main issues that should be considered towards the formal specification of a (maybe restricted) set of parallel basic linear algebra subprograms.