# A Square Block Format for Symmetric Band Matrices*

Fred G. Gustavson[1], José R. Herrero[2], E. Morancho[2]

[1]IBM T.J. Watson Research Center, Emeritus, and Umeå University
fg2935@hotmail.com

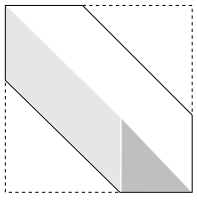
[2]Computer Architecture Department
Universitat Politècnica de Catalunya, BarcelonaTech
Barcelona, Spain
{josepr,enricm}@ac.upc.edu

**Abstract.** This contribution describes a Square Block, SB, format for storing a banded symmetric matrix. This is possible by rearranging "in place" LAPACK Band Layout to become a SB layout: store submatrices as a set of square blocks. The new format reduces storage space, provides higher locality of memory accesses, results in regular access patterns, and exposes parallelism.

## 1 Introduction

A banded matrix $A$ can be stored as a dense matrix (Fig. 1a). However, this implies the storage of null elements outside of the band. LAPACK [1] specifies a format for storing a band matrix using a rectangular array $AB$. The elements outside of the band are not stored. We consider the case where the matrix is symmetric. Thus, we only need to store either the lower or the upper part. In this paper we consider the former case, i.e. uplo = 'L' (Fig. 1b).

1a. Dense Storage



1b. LAPACK Storage

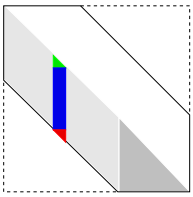
**Fig. 1.** Band matrix Storage in Dense (left) and LAPACK (right) formats

.

* This work was supported by the Spanish Ministry of Science and Technology (TIN2012-34557) and the Generalitat de Catalunya, Dep. d'Innovació, Universitats i Empresa (2009 SGR980).

In Dense storage, value $A_{i,j}$ is referenced in the code as $A(i,j)$. Thus, the $j^{th}$ diagonal element is stored in $A(j,j)$. In LAPACK Lower Band storage, uplo = 'L', the j-th column of $A$ is stored in the j-th column of $AB$ such that the diagonal element $A(j,j)$ is stored in $AB(1,j)$. Consequently, in LAPACK lower band codes $A_{i,j}$ is referenced as $AB_{1+i-j,j}$. This means that the correspondence is written in the code as $AB(1+i-j,j) = A(i,j)$ for $j <= i <= min(n, j+kd)$. This makes the code less readable than it could be. Figure 2 highlights the details of the storage of a *panel*, a set of contiguous columns.

2a. Dense Storage



2b. LAPACK Storage

**Fig. 2.** Storage of a Panel within the Band in Dense (left) and LAPACK (right) formats .

However, when $kd$, the *half bandwidth*, is small, the difference between dense and LAPACK storage requirements can be very large, clearly in favor of LAPACK storage. For a symmetric matrix of dimension $n$ and half bandwidth $kd$, dense storage requires the storage of $n^2$ values. Using LAPACK storage the banded matrix is stored in a rectangle containing $kd+1$ by $n$ values. The rectangle holds a parallelogram $P$ of dimension $kd+1$ by $n-kd$; and an isosceles triangle $T$ of side equal to $kd$. The rightmost white isosceles triangle seen in figure 1b within the rectangle corresponds to data allocated but not used. Clearly, this storage scheme incurs in space overhead, wasting about half the storage allocated to $T$.

### 1.1 Goals

Our goals include improving programmability and reducing storage requirements when operating on banded symmetric matrices. At the same time, we improve data locality and make parallelization more efficient. To do so, we are willing to rearrange the data so that:

- Space requirements are close to the optimum
- No further data copies or transformations are necessary at computation time
- Data management is more efficient
- Parallelization is easier and more efficient

### 1.2 Related Work

**Improved Programmability**
In [2] the authors describe a minor data format change for storing a symmetric band matrix $AB$ using the same array space specified by LAPACK [1]. In LAPACK lower band codes $AB_{i,j}$ is referenced in its code as $AB_{i-j+1,j}$. This makes

the code less readable than it could be as one would like to reference the $(i, j)$ element of a matrix $AB$ as $AB_{i,j}$. Furthermore, the layout of lower $AB$ in the LAPACK's user Guide, page 142 of [1] shows the user a rectangular matrix with the diagonal of $AB$ residing in the first row. Clearly, a layout description where the diagonal of $AB$ resides on the main diagonal of $AB$, see again page 142 of [1], is more suggestive and other things being equal is preferable. In [2] the authors improve Programmability of LAPACK Lower Band Cholesky by changing the Leading Dimension of $AB$ from $LDAB$ to $LDAB - 1$ in the array declaration of $AB$. This tells the compiler that the distance in the $2^{nd}$ dimension is one less. As a result one can write $AB(i, j)$ to access value $A(i, j)$. Also, in the layout description of $AB$ the diagonal of $AB$ is depicted as laying on the diagonal of $AB$.

**Improved Data Locality and Parallelization**

In [3] the authors propose Lower Blocked Column Packed and Upper Square Blocked Packed Formats, also known as Lower and Upper Block Packed Format (BPF) respectively. Both versions of BPF are alternatives to the Packed storage of a matrix used traditionally to conserve storage when that matrix has special properties. Two examples are symmetric and triangular matrices. By using BPF we may partition a symmetric matrix where each submatrix block is held contiguously in memory. This gives another way to pack a symmetric matrix and it avoids the data copies, that are inevitable when Level-3 BLAS are applied to matrices held in standard **C**olumn **M**ajor (CM) or **R**ow **M**ajor (RM) format as well as in standard packed format.

3a. Lower Blocked Packed Format

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | | | | | | |
| 1 | 9 | | | | | |
| 2 | 10 | 16 | | | | |
| 3 | 11 | 17 | 23 | | | |
| 4 | 12 | 18 | 24 | 28 | | |
| 5 | 13 | 19 | 25 | 29 | 33 | |
| 6 | 14 | 20 | 26 | 30 | 34 | 36 |
| 7 | 15 | 21 | 27 | 31 | 35 | 37 39 |

3b. Upper Blocked Packed Format

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 2 | 4 | 6 | 8 | 10 | 12 14 |
| | 3 | 5 | 7 | 9 | 11 | 13 15 |
| | | 16 | 18 | 20 | 22 | 24 26 |
| | | | 19 | 21 | 23 | 25 27 |
| | | | | 28 | 30 | 32 34 |
| | | | | | 31 | 33 35 |
| | | | | | | 36 38 |
| | | | | | | 39 |

**Fig. 3.** Lower and Upper Blocked Packed Formats of a Triangular Matrix

We define *lower* and *upper* BPF via an example in Fig. 3 with varying length rectangles of width $nb = 2$ and SB of order $nb = 2$ superimposed. Fig. 3 gives the memory addresses of the array that holds the matrix elements of BPF. The rectangles making up the array of Fig. 3 are in standard Fortran format and hence BPF supports calls to level-3 BLAS. The rectangles in Fig. 3a are *not* further divided into SB as these SB are *not* contiguous. Fig. 3 is a collection of $N = \lceil n/nb \rceil$ rectangular matrices concatenated together. The rectangles in Fig. 3b are the transposes of the rectangles in Fig. 3a and vice versa. Fig. 3b

rectangles have a *major* advantage over the rectangles of Fig. 3a: the $i^{th}$ rectangle consists of $N-i$ order $nb$ SB. This gives two dimensional contiguous granularity for _GEMM calls using upper BPF which lower BPF *cannot* possess. Lower BPF is *not* a preferred format over upper BPF as it does not give rise to contiguous SB. Another advantage of using upper BPF is one may at factor stage $i$ call _GEMM $(N-i-1)(i-1)$ times where each call is a parallel SB _GEMM update. This approach was used by LAPACK multicore Cholesky implementations [4, 5] among others. This implies that a BPF layout supports both traditional and multicore LAPACK implementations. Upper BPF is the preferred format. For further details see [3] and the references therein.

## 2 Upper Square Block Band Format

We could store a band matrix using Upper BPF (see figure 4). If we did so, we would be unnecessarily storing elements marked with * in the figure.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 4 | 6 | * | * | * | * |
| * | 3 | 5 | 7 | 9 | * | * | * |
| | | 16 | 18 | 20 | 22 | * | * |
| | | * | 19 | 21 | 23 | 25 | * |
| | | | | 28 | 30 | 32 | 34 |
| | | | | * | 31 | 33 | 35 |
| | | | | | | 36 | 38 |
| | | | | | | * | 39 |

**Fig. 4.** A band matrix stored in Upper Blocked Packed Format

Trying to reduce the unused space we can avoid storing those SB which only store null elements outside of the band. With this we could reduce the storage considerably. In the example in figure 4 we could avoid the storage of the top rightmost SB. However, we want to reduce further the storage of the part of the matrix which stores the parallelogram $P$. Let us observe in figure 4 the blocks which keep the boundaries of the band. Blocks which keep the main diagonal store a lower triangle L which is not used. For instance, for the blocks of size $nb = 2$ in figure 5 we can observe that memory address 1 is not used. Similarly, blocks which include the outermost diagonal store an upper triangle U which is not used at all. We can only observe that memory address 9 is the only one being used within the block that contains it. From this observations we conclude that we could save space by storing in address 1 the value originally hold in address 9. The same could be done with the value in memory address 25 which could be stored in the unused space in address 17. If we did so, the blocks containing the values in memory addresses 9 and 25 would not be needed. In general, in each block row holding a slab of $P$ we could avoid storing the lower triangle originally stored in the rightmost non-null block by storing it in the unused space corresponding to the lower triangle of the leftmost block in that slab. We

refer to this new format as Upper Square Block Band Format (USBBF). The final triangular part $T$ can just be stored in Upper BPF, which is compatible with USBBF.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 4 | 6 | ∗ | ∗ | ∗ | ∗ |
| 1 | 3 | 5 | 7 | 9 | ∗ | ∗ | ∗ |
| | | 16 | 18 | 20 | 22 | ∗ | ∗ |
| | | 17 | 19 | 21 | 23 | 25 | ∗ |
| | | | | 28 | 30 | 32 | 34 |
| | | | | ∗ | 31 | 33 | 35 |
| | | | | | | 36 | 38 |
| | | | | | | ∗ | 39 |

**Fig. 5.** A band matrix stored in Upper Blocked Packed Format

### 2.1 Data Transformation Process

Fig. 6[1] shows graphically the transformation process from a panel within the band ($P$ part) stored in LAPACK format into a slab in USBBF. The panel needs to be transposed and the bordering triangles joined in a single block.

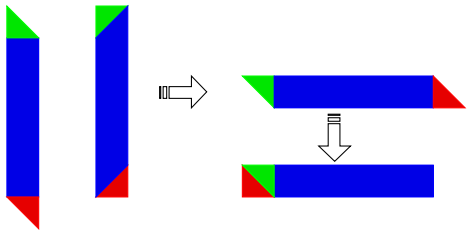


**Fig. 6.** From LAPACK Lower Band Format into Upper Square Block Band Format

It is possible to perform these data transformations fast in-place based on the work published in [6] and [7]. The process implies partitioning the matrix into submatrices and transposing them. This is achieved with a series of Shuffle/Unshuffle, and Transposition operations described in [7].

### 2.2 Final Layout

The new layout for uplo = 'L' consists of two geometric figures; a parallelogram $P$ and a lower isosceles triangle $T$ of side equal to $kd$. $P$ and $T$ must be stored in compatible formats:

[1] Readers can get a color version of the figures via email to the authors.

– P is stored in Upper Square Block Band Format (USBBF)
– T is stored in Upper Square Block Packed Format (Upper BPF)

The final layout stores $P$ and $T$ as shown in Fig. 7.

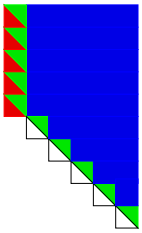


**Fig. 7.** Final Layout of a Band Matrix transformed to USBBF.

We must note that $kd$ is arbitrary while $nb$ is not. This means the boundary between the band ending and the blocked $T$ beginning is not necessarily on a multiple of $nb$ as Fig. 7 suggests. This issue can be handled in general. However, for clarity of presentation we make the simplifying assumption that $kd+1$ is a multiple of $nb$. Then Fig. 7 is accurate. This also eases the programming effort.

The parallelogram is partitioned into slabs of width $nb$. Each slab of $P$ is also a parallelogram $P_i$ of size $kd+1$ by $nb$. $P_i$ consists of two isosceles triangles of sizes $nb$ and $nb-1$ and a rectangle $R_i$ of size $kd+1-nb$ by $nb$. Now the two triangles concatenate to form a SB of order $nb$. Hence, $P_i$ also consists of just a SB and $R_i$. By transposing $R_i$ in-place $R_i$ becomes $\lfloor (kd+1)/nb-1 \rfloor$ SB's plus a leftover rectangular block. We note that transposing $AB$ gives an uplo = 'U' LAPACK implementation starting from the uplo = 'L' implementation. Thus, to get our SB formulation we follow this procedure. Triangle $T$ now becomes an upper isosceles triangle. We also map $T$ into upper blocked packed format [3] so it becomes "compatible" with the transposed parallelogram $P$.

The band in $P$ can be stored with minimal storage. Using full format to store the final triangle $T$ as in LAPACK requires that $\texttt{LDA} \geq KD+1$. Clearly, this wastes about half the storage allocated by Fortran or C to $T$. On the other hand, for each SB, $\texttt{LDA} = nb$. This means *minimal* storage is wasted for large $KD$ when $T$ is stored in Upper BPF. Therefore, this implies space savings w.r.t. LAPACK band storage.

## 3 Ongoing work

We are currently implementing an optimized parallel band Cholesky factorization based on USBBF. As we have shown in this paper, the new format stores submatrices as a set of square blocks. This provides higher locality of memory accesses, results in regular access patterns, and exposes parallelism. Consequently, this allows for efficient execution of kernels working on square blocks in parallel.

- No further data copies or transformations are necessary at computation time;
- Data off-loading is more efficient;
- Can use regular BLAS or LAPACK codes, or Specialized kernels [8, 9];
- Can be parallelized more easily with Dynamic Task Scheduling based on a Task Dependency Graph [10].

## 4 Conclusions

The new Upper Square Block Band Format (USBBF) stores submatrices as a set of square blocks. This reduces storage space, provides higher locality of memory accesses, results in regular access patterns, and exposes parallelism. The data transformation can be done very efficiently in-place and in parallel.

## References


1. Anderson, E., Bai, Z., Bischof, C., Blackford, L.S., Demmel, J., Dongarra, J.J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide. Third edn. Society for Industrial and Applied Mathematics, Philadelphia, PA (1999)
2. Gustavson, F.G., Quintana-Ortí, E.S., Quintana-Ortí, G., Remón, A., Waśniewski, J.: Clearer, Simpler and more Efficient LAPACK Routines for Symmetric Positive Definite Band Factorization. In: PARA'08. (May 2008)
3. Gustavson, F.G., Waśniewski, J., Dongarra, J.J., Herrero, J.R., Langou, J.: Level-3 Cholesky factorization routines improve performance of many Cholesky algorithms. ACM Transactions on Mathematical Software **39**(2) (February 2013) 9:1–9:10
4. Kurzak, J., Buttari, A., Dongarra, J.: Solving Systems of Linear Equations on the Cell Processor using Cholesky Factorization. IEEE Trans. Parallel Distrib. Syst. **19,9** (2008) 1175–1186
5. Quintana-Ortí, G., Quintana-Ortí, E.S., Remón, A., Geijn, R.A.: An Algorithm-by-Blocks for SuperMatrix Band Cholesky Factorization. In Palma, J.M., Amestoy, P.R., Daydé, M., Mattoso, M., Lopes, J.a.C., eds.: High Performance Computing for Computational Science - VECPAR 2008, Berlin, Heidelberg, Springer-Verlag (2008) 228–239
6. Gustavson, F.G., Karlsson, L., Kågström, B.: Parallel and Cache-Efficient In-Place Matrix Storage Format Conversion. ACM TOMS **38**(3) (April 2012) 17:1–17:32
7. Gustavson, F.G., Walker, D.: Algorithms for In-Place Matrix Transposition. In: Int. Conf. on Parallel Processing and Applied Mathematics (PPAM'13). Volume This Volume of Lecture Notes in Computer Science. (September 2013)
8. Herrero, J.R., Navarro, J.J.: Compiler-optimized kernels: An efficient alternative to hand-coded inner kernels. In: Proceedings of the International Conference on Computational Science and its Applications (ICCSA). LNCS 3984. (May 2006) 762–771
9. Herrero, J.R.: New data structures for matrices and specialized inner kernels: Low overhead for high performance. In: Int. Conf. on Parallel Processing and Applied Mathematics (PPAM'07). Volume 4967 of Lecture Notes in Computer Science. (September 2007) 659–667
10. Herrero, J.R.: Exposing Inner Kernels and Block Storage for Fast Parallel Dense Linear Algebra Codes. In: PARA'08. (May 2008)