Three Algorithms for Cholesky Factorization on Distributed Memory using Packed Storage

Fred G. Gustavson[1,2], Lars Karlsson[2], and Bo Kågström[2]

[1] IBM's T. J. Watson Research Center,
Yorktown Heights, NY 10598, USA, fg2@us.ibm.com
[2] Department of Computing Science and HPC2N, Umeå University,
S-901 87 Umeå, Sweden, {larsk, bokg}@cs.umu.se

Abstract. We present three algorithms for Cholesky factorization using minimum block storage for a distributed memory (DM) environment. One of the distributed square blocked packed (SBP) format algorithms performs similar to ScaLAPACK `PDPOTRF`, and with iteration overlapping outperforms it by as much as 67%. By storing the blocks in a standard contiguous way, we get better performing BLAS operations. Our DM algorithms are almost insensitive to memory hierarchy effects and thus gives smooth and predictable performance. We investigate the intricacies of using RFP format in a DM ScaLAPACK environment and point out some advantages and drawbacks.

1 Near Minimal Storage in a Serial Environment

Rectangular full packed (RFP) format is a standard full storage two-dimensional array for triangular or symmetric matrices requiring minimum storage [3]. For the lower triangular case, blocks A_{11}, A_{21}, A_{22}^T are stored as submatrices in a rectangular full storage array. This allows for using level 3 BLAS as well as making it easy to write LAPACK-style code for this format [3].

SBP format is a generalization of standard full storage. The matrix is partitioned into square blocks of order `NB`, and in the case of storing symmetric or triangular matrices, only the triangular *blocks* are stored. Each square block is contiguous in memory; the blocks are either stored row or column-wise. Each square diagonal block wastes `NB(NB-1)/2` elements, a total of `N(NB-1)/2` elements summed over all `N/NB` diagonal blocks. Each square block will map into L1 cache in an optimal way resulting in efficient BLAS operations.

2 Minimum Block Storage in a Distributed Environment

The current industry standard for distributed memory computing views the processors as a `PxQ` mesh and uses a 2D Block Cyclic Layout (BCL) of full format arrays. This has proven to be a good choice for achieving effective load balancing. However, this wastes about half the storage for triangular and symmetric matrices. There is currently no industry standard for packed storage. The SBP storage of Section 1 is a possibility.

2.1 Two Distributed RFP Algorithms for Cholesky Factorization

Both algorithms use the RFP layout [3]. One of these, due to lack of space, we do not discuss; see [3]. The other, a ScaLAPACK type algorithm, has the same form as one of the four RFP LAPACK algorithms in [3]. However, ScaLAPACK does not support SBP format. Nonetheless, RFP format can be distributed in a BCL and hence ScaLAPACK codes can be successfully used on it. This provides code reuse and portability. However, `PDPOTRF` and PBLAS have alignment restrictions. We found two solutions around this: runtime realignment or using more memory to store the misaligned submatrices. We have implemented one variant and experienced bad performance. We found two sources for this: more communication stages and less efficient BLAS operations on the nodes.

2.2 A Distributed SBP Algorithm for Cholesky Factorization

The square blocks in SBP can be considered atomic units. By using a BCL with blocking parameter `NB`, the square blocks are assigned to the mesh of processors. Instead of storing all blocks, we only store the triangular blocks, resulting in minimum block storage. Each block column is stored contiguously in local memory and we introduce a column pointer (CP) array that describes where each block column starts in the one dimensional array of local blocks. This CP array enables efficient referencing of the local blocks. Since each block is stored

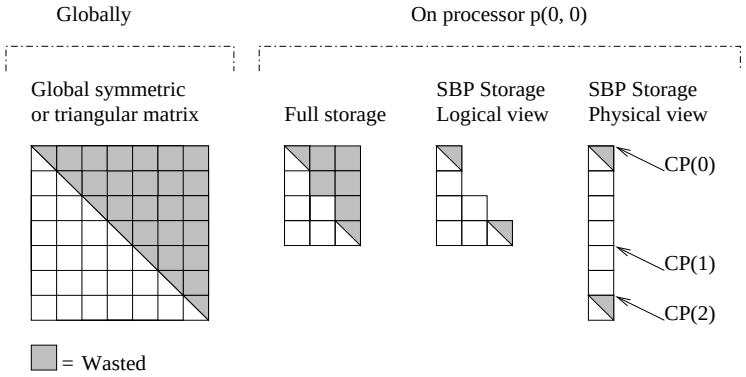

Fig. 1. Illustration of how an order 7 block global matrix is laid out on a 2×3 mesh in SBP format and addressed with its column pointer (CP) array.

contiguously BLAS operations involving them will achieve very good performance [3]. We show how right looking Cholesky factorization can be viewed as a series of pivot panel factorizations and Schur complement updates (SCUs). The SCU stage of each iteration requires row and column broadcasts of the associated pivot panel [3]. We show how this can be performed in two ways. The first approach is similar to ScaLAPACK Cholesky factorization and consists of a

series of computation and communication stages, resulting in a great deal of idle time because panel factorization only involves one panel of one process column.

By observing that panel factorization can be performed prior to the completion of the previous SCU stage, it is possible to overlap the factorization of the next panel by the SCU update of the current panel; see reference [1] of [3]. This approach is an extension to the algorithm presented in 1993 by Dackland et al [4] to a 2D BCL. We verify with execution tracing that the idle time caused by panel factorization is eliminated, and we present performance measurements suggesting a $11 - 67\%$ increase in MFLOPs for matrix orders up to 30000 compared with ScaLAPACK `PDPOTRF`. The increase in performance diminishes with increasing problem size since the asymptotically dominant operation is the SCU. Figure 2 illustrates how two sets of border vectors holding pivot panels can be

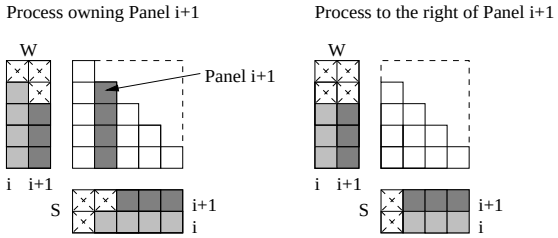

Fig. 2. Data layout for the SBP with double sets of W and S border vectors.

used to perform SCU i whilst at the same time performing a communication algorithm on the panel for iteration $i + 1$.

Experiments confirm that performance is smooth without spikes due to memory hierarchy effects. This is in contrast to the ScaLAPACK algorithm in which these effects are due to bad choices of local leading dimension. Table 1 shows

Table 1. Execution time for `PDPOTRF` and the SBP algorithm with iteration overlap for various square grid sizes.

N	2x2	3x3	4x4	5x5	6x6	7x7
4000	2.13/0.86	1.48/0.63	1.04/0.66	0.79/0.68	0.63/0.64	0.57/0.65
8000	14.80/0.92	8.29/0.80	5.33/0.79	3.97/0.77	3.15/0.71	2.64/0.73
12000		25.20/0.83	16.30/0.80	10.90/0.84	8.27/0.80	7.11/0.78
16000		57.30/0.84	34.50/0.85	24.00/0.85	18.30/0.80	13.90/0.85
20000			65.00/0.85	43.90/0.86	33.00/0.81	25.90/0.84
24000					53.90/0.84	42.30/0.85

selected times for both `PDPOTRF` and the algorithm using SBP and iteration overlapping. Each cell has the form X/y, where X is the time (in seconds) of the `PDPOTRF` routine and y = Y/X, where Y is the time for the SBP algorithm.

3 Related Work on DM Cholesky Factorization

We briefly discuss other packed storage schemes for DM environments. D'Azevedo and Dongarra suggested in 1997 a 1D storage scheme where each block column is stored in standard format [1] Benefits include code reuse and ease of use via PBLAS and ScaLAPACK routines. However, some new PBLAS routines seem to be required to handle the packed storage [1]. Furthermore, their results indicate that the performance varies wildly with input, making performance extrapolation difficult.

Recently, Marc Baboulin et al. presented a storage scheme which uses relatively large square blocks consisting of at least LCM(p, q) elementary blocks [2]. This format also supports code reuse via PBLAS and ScaLAPACK. The granularity is limited to the distributed block size, which means less possibility to save memory. For the Cholesky factorization routines, the chosen block sizes for performance measurements were between 1024 and 10240. This resulted in a departure from their minimal storage by as much as 7–13%. Using their minimal allowed distributed block size would bring this percentage down to about 1–3%. Our algorithms use the minimum block storage.

4 Conclusions

We have studied two algorithms and data formats for minimal block storage in distributed memory environments using a 2D block cyclic data layout.

In a serial environment, the RFP format is an attractive choice. However, combined with a ScaLAPACK layout, we have identified and shown a number of problems and weaknesses.

The SBP format was implemented and tested with two algorithm variants. One resembles ScaLAPACK's `PDPOTRF` but makes no use of PBLAS or ScaLAPACK routines, and one which overlaps iterations. We have demonstrated that performance at least as good as the ScaLAPACK algorithm is attainable, and for the overlapping variant achieving far better performance, especially for small and medium-sized matrices.

References

1. D'Azevedo, E., Dongarra, J.: Packed Storage Extension for ScaLAPACK. Technical Report, Oak Ridge National Laboratory (1997).
2. Baboulin, M., Giraud, L., Gratton, S., Langou, J.: A Distributed Packed Storage for Large Parallel Calculations. Technical Report, CERFACS (2005).
3. Gustavson, F.: New Generalized Data Structures for Matrices Lead to a Variety of High Performance Dense Linear Algebra Algorithms. In J. Dongarra, K. Madsen, and J. Wasniewski (editors), *PARA 2004, Applied Parallel Computing, State of the Art in Scientific Computing*, LNCS 3732, 2006, pp 11–20.
4. Dackland, K., Elmroth, E., Kågström, B.: A Ring–Oriented Approach for Block Matrix Factorizations on Shared and Distributed Memory Architectures. In R.F. Sincovec et al (editors), *Proc. Sixth SIAM Conf. on Parallel Processing for Scientific Computing*, SIAM Publications, 1993, pp 330–338.