Ring-oriented Block Matrix Factorization Algorithms for Shared and Distributed Memory Architectures

Krister Dackland and Erik Elmroth
Institute of Information Processing
University of Umeå
S-901 87 Umeå, Sweden

June 3, 1992
UMINF-92.04
ISSN-0348-0542

Abstract

Utilizing experiences from the implementations on shared memory multiprocessors (SMM) and distributed memory multicomputers (DMM), general ring-oriented routines are developed for the *LU*, *Cholesky*, and *QR* factorizations. Since, all machine dependencies are comprised to a small set of communication routines, the same factorization routines can be used on both the SMM and DMM architectures.

The algorithms are described on high level with focus on the portability aspects. Further, detailed implementations of the *LU* factorization and machine specific communication routines for the Alliant FX2816, Intel iPSC/2, and IBM 3090VF/600J are enclosed. Timing results show that the performance of machine specific implementations are preserved for the general ring-oriented block algorithms.

Keywords: Block matrix factorizations, parallel algorithms, portability, shared and distributed memory architectures.

1 Introduction

With the introduction of advanced parallel computer architectures a demand for efficient and portable algorithms has emerged. Several attempts concerning the possibilities to design algorithms and implementations that are portable between *shared memory multiprocessors* (SMM) and *distributed memory multicomputers* (DMM) have been made. For example, implementations of virtual shared memory on DMM architectures [6, 16, 23, 25] and the simulation of message passing with portable communication libraries on SMM architectures [5, 6, 15]. A major drawback with these attempts is that the demands for generality often deteriorates the performance for many problems that can be efficiently solved if the portability aspects are neglected. However, it is possible that the portability can be obtained without loss of performance for a restricted class of problems. In this paper the class of problems is restricted to parallel block matrix factorizations, such as the *LU*, *Cholesky*, and *QR* factorizations [7, 8, 9, 11, 13, 14]. This class of problems can easily be enlarged to most block algorithms, such as the ones included in LAPACK [3].

The goal for this contribution is to construct algorithms that, without loss of performance compared to implementations tuned for specific machines, are portable over and between different SMM and DMM architectures. This is done through further development of the ring-oriented block algorithms presented for both DMM and SMM architectures in [7, 8, 9].

The algorithms are described on high level with focus on the portability aspects. Results are presented for implementations on Intel iPSC/2 hypercube, Alliant FX2816, and IBM 3090 VF/600J and compared to the machine specific results presented in [7, 9].

The outline is as follows: a ring-oriented block algorithm is presented in the perspective of a DMM architecture in Section 2. The algorithm is adapted to a SMM architecture in Section 3, followed by a general ring-oriented block algorithm in Section 4. Performance results are presented in Section 5 and discussed in Section 6. Finally, in Section 7, some concluding remarks are summarized.

2 Block Ring Algorithms

In [7] efficient block right looking algorithms for the *LU*, *Cholesky*, and *QR* factorizations on a DMM architecture are presented. The connection topology for the DMM is a unidirected ring and the matrix A to be factorized is block column wrap mapped onto the nodes to achieve good load balancing. For a given processor i $(= 0 : p-1)$, these blocks form a local matrix A_{local}, comprising the column blocks $i+1$, $i+1+p$, $i+1+2p$, etc of A, where p is the number of processors.

On a high level the three algorithms may be described in a similar way: One processor, denoted *current processor* factorizes one block column and sends the factor, denoted *current block* to all other processors. With respect to the current block, all processors update their *remaining blocks*, i.e. the blocks to the right of the last factorized block column in A. The next current processor is the one to the right of the present one in the ring. The next current processor which holds the next block to be factorized, updates and factorizes that block and sends the next current block to all processors before it completes the update corresponding to the previous factorization. This enables an overlapping called *pipelining between iterations* in [7, 9].

In the implementations of the algorithms, the factorization of a block column is performed by a call to the corresponding level 2 routine, and the update is performed by one or more calls to level 3 BLAS [10, 21, 22] i.e. `DGEMM` and `DTRSM` in LU , `DGEMM`, `DSYRK`, and `DTRSM` in $Cholesky$, `DGEMM` and `DTRMM` in QR.

The only extra storage needed in these algorithms is a matrix to hold the current block, which mainly consists of the last factorized block column. All synchronizations are implicit through the message passing of the current blocks.

The only data, except for the current blocks, a processor refers to is A_{local}. This gives an inherited data locality from the initial matrix distribution in a DMM environment. Since each processor performs the work on separate block columns, this may also give good data locality for these algorithms applied to a SMM environment.

3 Shared Memory and Block Ring Algorithms

In this section we describe how to make an efficient implementation of the algorithm described in Section 2 on a SMM architecture. The idea is to use the DMM paradigm as a model and run exactly the same factorization routines on a SMM architecture viewed as a ring of processors, see Figure 3.1.

To accomplish this we reorganize the matrix A to be factorized as a ma-

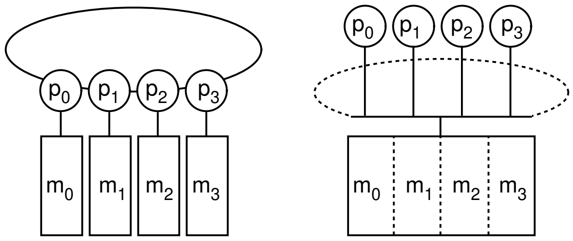

Figure 3.1: DMM and SMM models viewed as a ring of processors

trix A_{reorg} where the block columns of A are interchanged in a block column wrap mapped manner so that the blocks to be factorized by a given processor are placed consecutively. This gives each processor a logical local matrix A_{local} that refers to a submatrix of the restructured global matrix A_{reorg}. From this construction follows that the references to A_{local} are identical to the ones in the DMM algorithm described in section 2.

For use of the DMM algorithms on a SMM architecture, some considerations must be made for the current block that mainly consists of a factorized block column of A. In the DMM algorithm this block is sent from one processor to all other that keep it as a local copy. In the SMM model the current block is a block column of A_{reorg} in the common shared memory, which can be accessed by all processors.

The message passing of the current block is implemented as a change of reference for the current block to different locations in A_{reorg}. When a new block column is factorized a send is simulated by the update of a *global current block pointer*. A reception is simulated by assigning the value of the global current block pointer to a *local current block pointer*. To be sure that the global current block pointer has the right value and is ready to be read the update of it is synchronized by semaphores. The communication primitives are implemented with the same interfaces as for the DMM environment.

4 A General Algorithm

Paying attention to the differences in implementations of matrix factorizations on SMM and DMM architectures mentioned in the previous sections, we here form an algorithm that can be implemented on both SMM and DMM architectures with all machine dependencies in the communication routines. In the following this kind of ring-oriented block algorithms will be referred to as PASDMA (Portable Algorithms for Shared and Distributed Memory Architectures).

Consider an $m \times n$ matrix A that on a DMM is mapped onto the processors as described in section 2 and on a SMM is reorganized as described in section 3. In both environments the matrix A_{local} refers to the same parts of A. Let *cb_index* denote the block column index for the current block. Then a high level node algorithm for computing a matrix factorization is described as:

For $i\_global = 1 : nbl$
  If (I hold block i_global)
    If ($i\_global > 1$)
      UPDATE $A_{local}(i)$ w.r.t *CURRENT_BLOCK*(*cb_index*) (1)
    End If
    FACTORIZE $A_{local}(i)$ & generate *NEXT_CURRENT_BLOCK* (2)
    BROADCAST(*NEXT_CURRENT_BLOCK*, *cb_index*) (3)
    i = i + 1
  End If
  If ($i\_global > 1$)
    UPDATE $A_{local}(i : nbl_{local})$ w.r.t. *CURRENT_BLOCK*(*cb_index*) (4)
  End If
  RECEIVE(*CURRENT_BLOCK*, *cb_index*) (5)
End For

The block index i is initialized to 1, nbl_{local} to the number of block columns in A_{local}, and nbl to the number of block columns in A (for clarity we assume $m \geq n$).

In the implementations of the algorithm, both A_{local} and *CURRENT_BLOCK* are passed as arguments to the routine. On a DMM environment the argument for *CURRENT_BLOCK* is a working space of the same size as one block column. On a SMM environment it is the matrix A_{reorg}.

In the DMM environment the message passing operations ((3) and (5)) are implemented as ordinary broadcast and receive routines. The routines broadcasts and receives the matrix $CURRENT\_BLOCK$ and return always the value 1 for cb_index.

In a SMM environment the $CURRENT\_BLOCK$ refers to the global matrix A_{reorg} and therefore the communication routines only changes the value of cb_index to the correct location in the matrix.

A PASDMA implementation in FORTRAN of the LU factorization with partial pivoting is presented in Appendix A. The implementations of the $Cholesky$ and QR factorizations are quite similar.

5 Performance results

In this section performance results are presented for the LU, $Cholesky$, and QR factorizations implemented as node subprograms in FORTRAN and compiled for three different architectures. All machine and compiler dependencies are restricted to the implementations of the message passing routines, making it possible to run exactly the same factorization subprograms on different architectures. When using these node subprograms the nodes are assumed to be allocated and the matrix to be reordered or distributed. When timing the routines only the execution time for the factorizations are measured.

For each machine, the communication routines used, are presented in Appendix B. The reason to show the routines is to indicate the amount of work needed to install PASDMA implementations on a specific machine. Notice that the implementations of the communication routines are made in FORTRAN, utilizing extensions for synchronization of parallel computations on the different target architectures. They are not general, but sufficient for the implementation of the LU, $Cholesky$, and QR factorizations. The routines needed are `DBCAST`, `DREC`, `IBCAST`, and `IREC` for broadcast and reception of double precision and integer matrices, respectively.

Performance results for the factorization routines are presented for Alliant FX2816, Intel iPSC/2, and IBM 3090 VF/600J in Sections 5.1, 5.2, and 5.3, respectively. The results in Tables 5.2 - 5.4 show the execution time $T(p)$ measured as the maximum over p nodes and the performance measured in *Mflops*, computed as $T(p)/\#flops$, where *# flops* is counted as in Table 5.1. The block size nb giving the best performance for a given number of processors is also displayed. Also the *parallel speedup* $S(p)$ and the *parallel efficiency* $E(p)$ are presented for the algorithms. The *parallel*

Routine	Number of floating point operations
LU	$mn^2 - \frac{n^3}{3} - \frac{n^2}{2} + \frac{5n}{6}$
Cholesky	$\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$
QR $(m \geq n)$	$2mn^2 - \frac{2n^3}{3} + mn + n^2 + \frac{14n}{3}$
$(m \leq n)$	$2nm^2 - \frac{2m^3}{3} + 3nm - m^2 + \frac{14n}{3}$

Table 5.1: Number of floating point operations in the *LU*, *Cholesky*, and QR factorizations.

speedup is computed as

$$S(p) = \frac{T(p)}{T_{p_{min}}},$$

where $T_{p_{min}}$ denotes the execution time on the minimum number of nodes that could solve the problem. The value of p_{min} is one for the Alliant and IBM systems while it is restricted by the available storage of each node for the iPSC/2 (at most 4 *Mbytes* per node including the code). The *parallel efficiency* is computed as

$$E(p) = \frac{S(p)}{p}.$$

In the Tables 5.3 and 5.4 that show results for the iPSC/2 and the IBM 3090, also the performance in *Mflops* for the best machine specific implementations in [7] and [9] are presented.

5.1 Alliant FX2816

The Alliant FX2816 is a SMM system consisting of 16 Intel i860 processors each with a theoretical peak performance of 40 *Mflops* in double precision real arithmetic. The machine has a hierarchical memory system comprising an 8 Kbytes on chip data cache, two cache memories of size 1 *Mbyte* each, and a global shared main storage of size 128 *Mbytes*.

Alliant provides a compiler, FX/FORTRAN-2800 version 1.2.0 [1], that has several parallelization primitives. The `CNCALL` makes it possible to perform parallel subroutine calls in a `DO-loop` on a cluster of processors. To synchronize message passing the semaphores `ENTER` and `LEAVE`, constructed from the Concentrix commands `lock` and `unlock`, are used. In the implementations the highly optimized assembler-coded BLAS DGEMM from the Alliant library libalgebra 2.0 [2] is used. The performance of DGEMM is about 35 *Mflops* on one processor. All other routines used are from the high performance and portable GEMM-based level 3 BLAS implemented in FORTRAN 77 [21, 22]. All times presented have been measured by the high resolution clock `HRCGET`.

The communication routines used in double precision are presented in Appendix B.1. On a SMM system, there is no need for explicit synchronization for the pivot vector in the LU factorization, since it is implicitly synchronized by the communication for the current block. Therefore, the communication routines for integer matrices are implemented as quick returns. Before calling a factorization routine, the message passing has to be initialized by a call to the initialization routine in Appendix B.1.

On the Alliant the choice of leading dimension is very important. For all timing results the value 1092 is used, which have shown to work well. In Table 5.2, performance results on one to eight processors for the three routines are shown. By restricting the size of the cluster to eight, only the two processors (of four) that have distinct cache controllers on each processor module, are used. This minimizes the memory conflicts in the global cache. The QR factorization shows the best performance followed by the LU and $Cholesky$ factorizations, respectively. Notably, all routines show parallel efficiency on $1 - 5$ processors similar to the results presented for IBM 3090VF/600J in [9].

5.2 Intel iPSC/2 Hypercube

The iPSC/2 [4] is a DMM system with 64 scalar SX nodes. Each node is equipped with an Intel 80386, 4 *Mbyte* memory, and a Weitec 1167, which has a theoretical peak performance just under 0.6 *Mflops* in double precision real arithmetic. The communication system uses a direct connect routing module (DCM), with a band-width of 2.8 *Mbytes/sec* in each direction. To perform the message passing we use the iPSC/2 system routines `CSEND` and `CRECV` and the execution times have been measured by `MCLOCK` [19]. The BLAS used are the standard f77 implementations available from Netlib [12] and compiled with the f77 compiler.

The communication routines used in double precision are presented in Appendix B.2. The communication routines for integer matrices are implemented similarly.

As described in Sections 2 - 4, the implementations of the PASDMA algorithms are modified versions of the ones described in [7]. Since the modifications mainly consist of changes in the parameter lists of the communication

PASDMA					
Routine	p	nb	*Mflops*	$S(p)$	$E(p)$
LU	1	64	25.1	1.00	1.00
	2	48	47.8	1.90	0.95
	3	48	67.2	2.67	0.89
	4	48	86.4	3.44	0.86
	5	48	102.0	4.06	0.81
	6	32	113.3	4.51	0.75
	7	32	125.7	5.01	0.72
	8	32	137.0	5.46	0.68
Cholesky	1	64	25.2	1.00	1.00
	2	64	47.7	1.89	0.95
	3	48	66.2	2.63	0.88
	4	32	79.8	3.17	0.79
	5	32	90.0	3.57	0.71
	6	32	96.0	3.81	0.63
	7	16	103.6	4.11	0.59
	8	16	109.5	4.35	0.54
QR	1	32	25.8	1.00	1.00
	2	32	49.4	1.92	0.96
	3	32	70.9	2.75	0.92
	4	32	90.8	3.52	0.88
	5	16	109.0	4.23	0.85
	6	16	125.5	4.87	0.81
	7	16	141.7	5.50	0.79
	8	16	153.2	5.95	0.74

Table 5.2: Results for the LU, $Cholesky$, and QR factorizations of a 1024×1024 matrix on Alliant FX2816.

PASDMA						Machine
Routine	p	nb	*Mflops*	$S(p)$	$E(p)$	Specific
LU	4	4	1.29	1.00	1.00	1.29
	8	2	2.53	1.96	0.98	2.51
	16	2	4.93	3.82	0.96	4.76
	32	2	9.40	7.29	0.91	8.71
	64	1	17.10	13.26	0.83	14.89
Cholesky	4	8	1.29	1.00	1.00	1.29
	8	8	2.54	1.97	0.99	2.53
	16	4	4.88	3.78	0.95	4.83
	32	2	8.73	6.77	0.85	8.39
	64	2	13.40	10.34	0.65	12.79
QR	4	4	1.55	1.00	1.00	1.55
	8	2	3.06	1.97	0.99	3.06
	16	2	5.99	3.86	0.97	6.00
	32	1	11.58	7.47	0.93	11.65
	64	2	21.54	13.90	0.87	21.71

Table 5.3: Results for the *LU*, *Cholesky*, and *QR* factorizations of a 1000×1000 matrix on Intel iPSC/2.

routines, the performance results shown in Table 5.3, are preserved from [7]. Furthermore, in the PASDMA implementation of the LU factorization the *swaps* are performed column-wise instead of row-wise, explaining the somewhat improved performance.

5.3 IBM 3090 VF/600J

The IBM 3090 VF/600J [27] is a SMM system with 6 processors and a a hierarchical memory system. All processors has a vector facility (VF) and the theoretical peak performance is 138 *Mflops* per processor in double precision real arithmetic. The practical peak performance obtained from the level-3 BLAS DGEMM is around 108 *Mflops* [9]. Besides 8 double precision vector registers per processor, the memory hierarchy consists of a cache memory of size 256 Kbytes per processor and a central storage of size 256 *Mbytes*. The `PARALLEL CALL` construction in VS FORTRAN 2.5 [18] is used to execute subroutines in parallel. The semaphores `PLLOCK` and `PLFREE` are used for synchronization of the message passing and `CLOCKX` to measure the execution time. All timing results have been produced on a semi-dedicated machine. The BLAS 3 used are from ESSL [17], except DTRMM that is from [20, 24].

The communication routines for the IBM 3090VF/600J are similar to the Alliant routines, except for the handling of the machine dependent semaphores. The double precision communication routines are presented in Appendix B.3.

In [9] performance results for machine specific parallel implementations of the LU, $Cholesky$, and QR factorizations are presented. In Table 5.4 these results are shown together with the results for the PASDMA implementations. It is here interesting to compare the results to see if the PASDMA concept is competable.

In the LU case the PASDMA variant is never more than 2.5 percentage from the machine specific one and on four processors the PASDMA variant is even faster. Notable is that the optimal block size nb is 32 for the PASDMA variant on $2-5$ processors, while it is 48 for the machine specific variant [9].

For the $Cholesky$ factorization the PASDMA variant is 8 percentage faster on one processor. This is explained by the use of a new assembler-coded DSYRK provided by ESSL. Further, the optimal block size nb is 256 instead of 32 as in [9]. On more than one processor the performances of the two variants are close, but the PASDMA variant is faster overall.

PASDMA						Machine
Routine	p	nb	*Mflops*	$S(p)$	$E(p)$	Specific
LU	1	48	97.3	1.00	1.00	98.1
	2	32	188.6	1.94	0.97	192.9
	3	32	275.7	2.83	0.94	277.6
	4	32	356.6	3.67	0.92	355.2
	5	32	427.9	4.40	0.88	431.0
Cholesky	1	256	99.6	1.00	1.00	91.9
	2	96	178.5	1.79	0.90	173.1
	3	64	255.0	2.56	0.85	247.2
	4	32	315.8	3.17	0.79	307.8
	5	48	353.8	3.55	0.71	354.4
QR	1	32	92.8	1.00	1.00	94.2
	2	32	183.2	1.97	0.99	182.9
	3	32	267.2	2.88	0.96	266.0
	4	32	345.8	3.73	0.93	345.2
	5	32	402.7	4.34	0.87	413.2

Table 5.4: Results for the *LU*, *Cholesky*, and *QR* factorizations of a 1200×1200 matrix on IBM 3090 VF/600J.

The performance for the QR factorization is also similar to the machine specific variant on $1-4$ processors. On 5 processors the PASDMA implementation is four percentages slower.

One of the reasons why the PASDMA variant sometimes is faster than the machine specific variant is that the implementations has been made in VS FORTRAN 2.5 instead of Parallel FORTRAN [26]. The parallel features of VS FORTRAN did not exist when the implementations in [9] were done.

6 Discussion

The timing results for the PASDMA implementations have shown performance similar to the machine specific implementations for both the Intel iPSC/2 and the IBM 3090 VF/600J.

The similarity in performance on the iPSC/2 is explained by the small differences between PASDMA implementations and the machine specific versions. From this follows that the modeling and performance evaluation in [7] also are valid for the PASDMA algorithms used on an iPSC/2.

For SMM environments the good performance of the PASDMA algorithms are mainly explained by the reorganization of the matrix, giving each processor block columns that are stored consecutively. The consecutive storage enables updating of several block columns in one operation, instead of one operation per block column. This implies less amount of software overhead and larger matrices in the level 3 BLAS operations. In [9], performance considerations, concerning software overhead and level 3 fractions for parallel block matrix factorizations on a SMM system, are discussed.

The PASDMA approach is applicable to block algorithms in general, making it possible to construct a complete library of parallel block algorithms. This enables use of routines in sequence, hopefully without redistribution or reorganization of the matrix between each operation. If a reorganization is needed on a SMM system, it is an easily parallelized $\mathcal{O}(n^2)$ operation. The distribution problem on a DMM environment is equally expensive for the PASDMA and the machine specific versions.

The PASDMA concept is further improved by the use of GEMM-based level 3 BLAS [21, 22]. As a consequence, the PASDMA approach gives performance close to corresponding machine specific implementations on both DMM and SMM architectures, if only there is a highly optimized uniprocessor DGEMM routine available.

Except for the DGEMM, all machine dependencies are restricted to the communication primitives. The implementation of the communication primitives for a SMM environment, which already in FORTRAN 77 is a simple task, is even more simplified when using a language, such as FORTRAN 90, which includes dynamic data structures.

7 Conclusions

The PASDMA (Portable Algorithms for Shared and Distributed Memory Architectures) concept for construction of ring-oriented block algorithms is presented. It is shown that through use of this concept, only one set of routines is needed to exploit the full potential of both SMM and DMM environments. PASDMA is based on a matrix reorganization on SMM's, enabling use of DMM ring-oriented block algorithms on both SMM and DMM, with all machine dependencies restricted to the message passing routines.

Performance results are shown for implementations of the LU, $Cholesky$, and QR factorizations on Alliant FX2816, Intel iPSC/2, and IBM 3090 VF/600J. The performance of the PASDMA implementations are competable with the corresponding machine specific implementations on Intel iPSC/2 [7] and IBM 3090 VF/600J [9].

We conclude that the PASDMA concept makes it possible to develop portable algorithms for both DMM and SMM environments that reach performance similar to the machine specific implementations.

Acknowledgements

We would like to thank our colleagues in the Parallel Computing Research Group at the University of Umeå for their support and especially our supervisor Bo Kågström for helpful discussions and constructive comments during the work and the preparation of this paper.

This work was partly supported by the Swedish Board of Technical Development under grant STU 89-02578P.

References

[1] Alliant Computer Systems Corporation "FX/FORTRAN-2800, Programmer's Handbook", 1990.

[2] Alliant Computer Systems Corporation "FX/SERIES Linear Algebra Library", 1990.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, *LAPACK Users' Guide*, SIAM Publications, (1992).

[4] S. Arshi, R. Asbury, J. Brandenburg, and D. Scott "Application Performance Improvement on the iPSC/2 Computer", Concurrent Supercomputing, The second Generation, A technical Summary of the iPSC/2 Concurrent Supercomputer, Intel, pp 17-22.

[5] L. Bomans, D. Roose, and R. Hempel, "The Argonne/GMD macros in Fortran for portable programming and their implementation on the Intel iPSC/2", *Parallel Computing*, 15, (1990), pp 119-132.

[6] J. Boyle, R. Butler, T. Disz, B. Glickfeld, E. Lusk, R. Overbeek, J. Patterson, and R. Stevens, *Portable Programs for Parallel Processing*, Holt, Rinehart and Winston, Inc, (1987).

[7] K. Dackland and E. Elmroth, "Parallel Block Matrix Factorizations for Distributed Memory Multicomputers", Report UMINF-92.03, Inst. of Information Processing, Univ. of Umeå, S-901 87 Umeå, May 1992.

[8] K. Dackland, E. Elmroth, B. Kågström, and C. Van Loan, "Design and Evaluation of Parallel Block Algorithms: LU Factorization on an IBM 3090 VF/600J", in D. Sorensen et al (eds.), *Parallel Processing for Scientific Computing*, SIAM Publications, (1991) (to appear).

[9] K. Dackland, E. Elmroth, B. Kågström, and C. Van Loan, "Parallel Block Matrix Factorizations on the Shared Memory Multiprocessor IBM 3090 VF/600J", *International Journal of Supercomputer Applications*, Vol 6.1, MIT Press (1992), pp 69-97

[10] J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff, "A Set of Level 3 Basic Linear Algebra Subprograms", *ACM Trans. on Mathematical Software*, Vol. 16 (1990) pp 1-17, 18-28.

[11] J. Dongarra, I. Duff, D. Sorensen and H. Van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM Publications, (1991).

[12] J. Dongarra and E. Grosse, "Distribution of mathematical software via electronic mail", *Commun. ACM* 30, 5, (1987), 403 - 407.

[13] K. Gallivan, R. Plemmons and A. Sameh, "Parallel Algorithms for Dense Linear Algebra Computations", *SIAM Review*, Vol. 32 (1990), pp 54-135.

[14] G. Golub and C. Van Loan, *Matrix Computations*, John Hopkins Press, 2nd edition, (1989).

[15] A. Geist, M. Heath, B. Peyton, and P. Worley, "PICL a Portable Instrumented Communication Library", Tech. Memorandum ORNL/TM-6150, Oak Ridge National Laboratory, 1990, pp 1213-1222.

[16] H. Hellwagner, "A Survey of Virtually Shared Memory Schemes", TUM-I9056, Institut für Informatik, Technische Universität München, 1990.

[17] IBM, *Engineering and Scientific Subroutine Library Guide and Reference*, SC23-0184-3, November 1988.

[18] IBM, *VS FORTRAN Version 2 Release 5 Language and Library Reference*, SC26-4221-6, December 1990.

[19] Intel Corporation, "iPSC/2 and iPSC/860, Programmer's Reference Manual", June 1990.

[20] B. Kågström and P. Ling, "Level 2 and 3 BLAS Routines for IBM 3090 VF: Implementation and Experiences", in J. Dongarra, I. Duff, P. Gaffney and S. McKee(eds), *Vector and Parallel Computing*, Ellis Horwood, 1989, pp 229-240.

[21] B. Kågström P. Ling, and C. Van Loan, "High Performance GEMM-Based Level-3 BLAS: Sample Routines for Double Precision Real Data", in M. Durand and F. El Dabaghi (eds.), *High Performance Computing II*, Elsevier Science Publishers B.V. (North-Holland), (1991), pp 269-281.

[22] B. Kågström and C. Van Loan, "GEMM-Based Level-3 BLAS", Tech. Report CTC91TR47, Cornell University, December 1989.

[23] K. Li, "IVY: A Shared Virtual Memory on Loosely Coupled Multiprocessors", Ph.D. Thesis, Yale University, (1986).

[24] P. Ling, "A Set of High Performance Level-3 BLAS Structured and Tuned for the IBM 3090 VF and Implemented in Fortran 77", Report UMINF-179.90, Inst. of Information Processing, Univ. of Umeå, S-901 87 Umeå, May 1990.

[25] M. Stumm and S. Zhou, "Algorithms Implementing Distributed Shared Memory", *IEEE Computer*, Vol. 23, No. 5, (1989), pp 54-64.

[26] L. Toomey, E. Plachy, R. Scarborough, R. Sahulka, J. Shaw, and A. Shannon, "IBM Parallel Fortran", *IBM Systems Journal*, Vol. 27 (1988) pp 416-435.

[27] S. Tucker, "The IBM 3090 System: An Overview", *IBM Systems Journal*, Vol. 25(1), (1986) pp 4-20.

A APPENDIX - PASDMA implementation of the LU factorization

```
      SUBROUTINE LU(M,N,A,LDA,NA,NB,CB,LDCB,IPIV,INFO,ME,P)
*
      IMPLICIT NONE
*
*     .. SCALAR ARGUMENTS ..
      INTEGER M, N, LDA, LDCB,NA, NB, INFO, ME, P
*     ..
*     .. ARRAY ARGUMENTS ..
      INTEGER          IPIV(LDA)
      DOUBLE PRECISION A(LDA,*), CB(LDCB, *)
*     ..
*
*  PURPOSE
*  =======
*
*  LU COMPUTES AN LU FACTORIZATION OF A GENERAL M-BY-N MATRIX A
*  USING PARTIAL PIVOTING WITH ROW INTERCHANGES.
*
*  THE FACTORIZATION HAS THE FORM
*     A = P * L * U
*  WHERE P IS A PERMUTATION MATRIX, L IS LOWER TRIANGULAR WITH UNIT
*  DIAGONAL ELEMENTS (LOWER TRAPEZOIDAL IF M > N), AND U IS UPPER
*  TRIANGULAR (UPPER TRAPEZOIDAL IF M < N).
*
*  ARGUMENTS
*  =========
*
*
*  M    (INPUT) INTEGER
*       THE NUMBER OF ROWS IN THE MATRIX A.
*
*  N    (INPUT) INTEGER
*       THE NUMBER OF COLUMNS IN THE MATRIX THAT IS DISTRIBUTED IN BLOCK COLUMN
*       WRAP MAPPING OF THE NUMBER OF PROCESSORS.
*
*  A    (INPUT/OUTPUT) DOUBLE PRECISION ARRAY, DIMENSION (LDA, N)
*       ON ENTRY, THE M BY NA MATRIX A THAT IS THIS PROCESSORS           E
*       PART OF THE GLOBAL MATRIX.
*
*  LDA  (INPUT) INTEGER
*       THE LEADING DIMENSION OF THE ARRAY A. LDA >= M.
*
*  NA   (INPUT) INTEGER
*       THE NUMBER OF COLUMNS IN A.
```

```
*
*  NB   (INPUT) INTEGER
*       THE NUMBER OF COLUMNS IN EACH BLOCK COLUMN.
*
*  CB   (WORKSPACE) DOUBLE PRECISION ARRAY, DIMENSION (M, NB)
*       USED TO HOLD THE CURRENT BLOCK (CURRENT FACTOR) USED IN
*       UPDATES.
*
*  LDCB  (INPUT) INTEGER
*       THE LEADING DIMENSION OF THE ARRAY CB. LDCB >= M.
*
*  IPIV (OUTPUT) INTEGER ARRAY, DIMENSION (M)
*       THE PIVOT INDICES. ROW I OF THE MATRIX WAS INTERCHANGED
*       WITH ROW IPIV(I).
*
*  INFO (OUTPUT) INTEGER
*       = 0: SUCCESSFUL EXIT
*       < 0: IF INFO = -K, THE K-TH ARGUMENT HAD AN ILLEGAL VALUE
*       > 0: IF INFO = K, U(K,K) IS EXACTLY ZERO. THE FACTORIZATION
*            HAS BEEN COMPLETED, BUT THE FACTOR U IS EXACTLY
*            SINGULAR, AND DIVISION BY ZERO WILL OCCUR IF IT IS USED
*            TO SOLVE A SYSTEM OF EQUATIONS.
*
*  ME   (INPUT) INTEGER
*       PROCESS IDENTITY IN THE INTERVAL [0, P-1].
*
*  P    (INPUT) INTEGER
*       THE TOTAL NUMBER OF PROCESSORS.
*
*  ===================================================================
*
*     .. PARAMETERS ..
      INTEGER NBMAX, MATRIX, PIVOTS
      PARAMETER (NBMAX = 30,MATRIX=1,PIVOTS=2)
*     ..
*     .. LOCAL SCALARS ..
      INTEGER NBL, E, EL, II, IP, PE ,RIGHTN, LBL, J,III
      INTEGER MM, S, SL, SSL, I, PROC, FROM, IINFO
      INTEGER CBI, TMP, LOCAL_CBI
*     ..
*     .. EXTERNAL FUNCTIONS ..
      INTEGER RIGHT, INIT_L_CBI
*     ..
*     .. EXTERNAL SUBROUTINES ..
      EXTERNAL RIGHT, DSWAP, DGEMM, DTRSM, DGETF2,DBCAST,DREC
      EXTERNAL DCOPY, IBCAST, IREC, GRAY
*     ..
```

```
*     .. INTRINSIC FUNCTIONS ..
      INTEGER MIN, MAX
*     ..
*     .. EXECUTABLE STATEMENTS ..
*
*     TEST THE INPUT PARAMETERS.
*
      CBI = 0
      INFO = 0
      IF (M .LT. 0) THEN
         INFO = -1
      ELSE IF (N .LT. 0) THEN
         INFO = -2
      ELSE IF (LDA .LT. MAX(1,M)) THEN
         INFO = -4
      ELSE IF (NA .LT. 0) THEN
         INFO = -5
      ELSE IF (NB .LT. 0 .OR. NB .GT. NBMAX) THEN
         INFO = -6
      ELSE IF (LDCB .LT. MAX(1,M)) THEN
         INFO = -8
      ELSE IF (P .LT. 0) THEN
         INFO = -12
      ELSE IF ((ME .LT. 0) .OR. (ME .GE. P)) THEN
         INFO = -11
      END IF
      IF (INFO .LT. 0) THEN
         CALL XERBLA('LU', -INFO)
         RETURN
      END IF
      LOCAL_CBI = INIT_L_CBI(ME,N,NB,P)
*
*     QUICK RETURN IF POSSIBLE.
*
      IF( M.EQ.0 .OR. N.EQ.0 )
     $   RETURN

      NBL = (MIN(M, N) + NB - 1)/NB
      IF (N .GT. NBL*NB) NBL = NBL + 1
      RIGHTN = RIGHT(ME,P)
      LBL = 0
      PROC = 0

*
      DO 10 I = 1,NBL
         S = (I-1)*NB + 1
         E = MIN(I*NB, M)
```

```
*
*        THE PROCESSOR THAT HOLDS THE CURRENT BLOCK COLUMN
*        TO BE FACTORIZED, UPDATES THAT BLOCK COLUMN WITH
*        RESPECT TO THE PREVIOUS FACTORIZATION (ITERATION I-1)
*        AND PERFORMS THE FACTORIZATION FOR THE ITERATION I.
*
         IF ((PROC .EQ. ME) .AND. (S .LE. M)) THEN
            SL = LBL*NB + 1
            EL = MIN((LBL+1)*NB, NA)
            LBL = LBL + 1
            IF (I .GT. 1) THEN
*
*               APPLY PREVIOUS INTERCHANGES TO CURRENT BLOCK.
*
                CALL DLASWP(EL-SL+1,A(1,SL),LDA,S-NB,S-1,IPIV,1)
*
*               PERFORM UPDATE OF NEXT BLOCK COLUMN TO BE FACTORIZED.
*
                CALL DTRSM('L','L','N','U',NB,EL-SL+1,1.D0,
     $                  CB(S-NB,CBI),LDCB,A(S-NB,SL),LDA)

                CALL DGEMM('N','N',M-S+1,EL-SL+1,NB,-1.D0,
     $                 CB(S,CBI),LDCB,A(S-NB,SL),LDA,1.D0,A(S,SL),LDA)

              ENDIF
*
*             PERFORM LEVEL-2 FACTORIZATION AND ADJUST PIVOT VECTOR.
*
              CALL DGETF2(M-S+1,EL-SL+1,A(S,SL),LDA,IPIV(S),IINFO)
              IF (IINFO .NE. 0 .AND. INFO .NE. 0) THEN
                 INFO = IINFO + S - 1
              END IF

              DO 30 II = S,E
                 IPIV(II) = (I-1)*NB + IPIV(II)
30            CONTINUE
*
*             SEND THE NEW FACTOR AND PIVOT VECTOR TO ALL PROCESSORS.
*
              CALL DBCAST(M-S+1,NB,A(S,SL),LDA,ME,P,LOCAL_CBI,MATRIX)
              CALL IBCAST(E-S+1,1,IPIV(S),LDA,ME,1,P,PIVOTS)
           ENDIF
*
*         UPDATE WITH RESPECT TO THE PREVIOUS FACTORIZATION
*         (ITERATION I-1). ALL PROCESSORS UPDATE THEIR
*         BLOCK COLUMNS TO THE RIGHT OF THE BLOCK THAT IS FACTORIZED
*         IN THIS ITERATION (I).
```

```
*
         IF (I .GT. 1) THEN
            SSL = LBL * NB + 1
            IF (NA .GE. SSL) THEN
                CALL DLASWP(NA-SSL+1,A(1,SSL),LDA,S-NB,
     $                MIN(S-1,M),IPIV,1)

                CALL DTRSM('L','L','N','U',MIN(NB,M-S+NB+1),
     $            NA-SSL+1,1.D0,CB(S-NB,CBI),LDCB,A(S-NB,SSL),LDA)
                IF (S .LE. M) THEN
                   CALL DGEMM('N','N',M-S+1,NA-SSL+1,NB,-1.D0,
     $                  CB(S,CBI),LDCB,A(S-NB,SSL),LDA,1.D0,
     $                  A(S,SSL),LDA)
                END IF
             END IF
          END IF
*
*        RECEIVE THE FACTOR IN CB AND PIVOT VECTOR BELONGING TO
*        THE CURRENT FACTORIZATION (ITERATION I).
*
         IF (S .LE. M) THEN
            CALL DREC(M-S+1,NB,CB(S,1),LDCB,A(S,SL),LDA,
     $                ME,PROC,CBI,MATRIX)
            CALL IREC(E-S+1,1,IPIV(S),LDA,ME,PROC,TMP,PIVOTS)
         ENDIF
         PROC = RIGHT(PROC,P)
10     CONTINUE
*
*     APPLY INTERCHANGES TO PREVIOUS BLOCKS
*
      SL = 1
      DO 50 S = (ME+1)*NB+1,N-1,P*NB
         CALL DLASWP(NB,A(1,SL),LDA,S,MIN(M,N),IPIV,1)
         SL = SL + NB
50    CONTINUE

      RETURN
      END
```

B APPENDIX - Communication routines

Communication routines for the Alliant FX2816, iPSC/2, and IBM 3090VF/600J, used in the PASDMA implementations of the *LU*, *Cholesky*, and *QR* factorizations. The routines needed are `DBCAST`, `DREC`, `IBCAST`, and `IREC` for broadcast and reception of double precision and integer matrices, respectively. Only the routines for double precision real data are shown.

B.1 Alliant FX2816

```
      SUBROUTINE DBCAST(M,N,A,LDA,ME,P,LOCAL_CBI,MSGID)
      INTEGER M, N, ME, P, MSGID, LDA, LOCAL_CBI
      DOUBLE PRECISION A(LDA,*)
*
*     SETS GLOBAL CURRENT BLOCK POINTER (SHDATA(0)) TO
*     THE BEGINNING OF THE NEXT CURRENT BLOCK.
*
      INTEGER SHDATA(0:1), CB
      COMMON /SHARED1/ SHDATA
      PARAMETER (CB = 1)

      IF (MSGID .EQ. CB) THEN
99       CALL ENTER()
         IF (SHDATA(1) .NE. 0) THEN
            CALL LEAVE()
            GOTO 99
         ELSE
            SHDATA(0) = LOCAL_CBI
            LOCAL_CBI = LOCAL_CBI + N
            SHDATA(1) = P
         ENDIF
         CALL LEAVE()
      ENDIF
      RETURN
      END
```

```
      SUBROUTINE DREC(M, N, A, LDA, B, LDB, ME, PROC, CBI, MSGID)
      INTEGER CBI, MSGID, ME, M, N, LDA, PROC, LDB
      DOUBLE PRECISION A(LDA, *), B(LDB,*)
*
*     SETS CBI TO GLOBAL CURRENT BLOCK POINTER (SHDATA(0)) AND
*     DECREMENTS SHDATA(1) BY ONE TO SHOW THAT THE MESSAGE IS RECEIVED.
*
      INTEGER SHDATA(0:1), CB
      COMMON /SHARED1/ SHDATA
      PARAMETER (CB = 1)

      IF (MSGID .EQ. CB) THEN
99       CALL ENTER()
         IF (SHDATA(0) .EQ. CBI) THEN
            CALL LEAVE()
            GOTO 99
         ELSE
            CBI = SHDATA(0)
            SHDATA(1) = SHDATA(1) - 1
         ENDIF
         CALL LEAVE()
      ENDIF
      RETURN
      END
```

```
      SUBROUTINE INITMESSPASS()
      INTEGER SHDATA(0:1)
      COMMON /SHARED1/ SHDATA

      SHDATA(0) = 0
      SHDATA(1) = 0
      RETURN
      END
```

B.2 iPSC/2

```
      SUBROUTINE DBCAST(M, N, A, LDA, ME, P, LOCAL_CBI, MSGID)
      INTEGER M, N, LDA, IDEST, MSGID, ME, P, LOCAL_CBI
      DOUBLE PRECISION A(LDA,*)
*
*     SEND MATRIX A TO ALL PROCESSORS
*
      INTEGER LEN, I, LDW, PID, MYPID
      PARAMETER (LDW = 1000*16, IDEST = -1)
      DOUBLE PRECISION WORK(LDW)

      PID = MYPID()
      LEN = 8 * M * N
      DO 10 I = 1, N
         CALL DCOPY(M,A(1,I),1,WORK((I-1)*M+1),1)
10    CONTINUE
      CALL CSEND(MSGID,WORK,LEN,IDEST,PID)
      RETURN
      END
```

```
      SUBROUTINE DREC(M, N, A, LDA, B, LDB, ME, PROC, CBI, MSGID)
      INTEGER M, N, LDA, ME, PROC, CBI, MSGID
      DOUBLE PRECISION A(LDA,*), B(LDB,*)
*
*     RECEIVE MATRIX A FROM PROCESSOR PROC.
*     IF ME IS PROC, THE MATRIX B IS COPIED INTO A
*
      INTEGER I, LEN, LDW
      PARAMETER (LDW = 1000*16)
      DOUBLE PRECISION WORK(LDW)

      IF (ME .EQ. PROC) THEN
         DO 10 I = 1,N
            CALL DCOPY(M, B(1,I), 1, A(1,I), 1)
10       CONTINUE
      ELSE
         LEN = M * N * 8
         CALL CRECV(MSGID, WORK, LEN)
         DO 30 I = 1,N
            CALL DCOPY(M, WORK((I-1)*M+1), 1, A(1,I), 1)
30       CONTINUE
      END IF
      CBI = 1
      RETURN
      END
```

B.3 IBM 3090VF/600J

```
      SUBROUTINE DBCAST(M,N,A,LDA,ME,P,LOCAL_CBI,MSGID)
      INTEGER M, N, ME, P, MSGID, LDA, LOCAL_CBI
      DOUBLE PRECISION A(LDA,*)
*
*     SETS GLOBAL CURRENT BLOCK POINTER (SHDATA(0)) TO
*     THE BEGINNING OF THE NEXT CURRENT BLOCK.
*
      INTEGER SHDATA(0:2), CB
      COMMON /SHARED1/ SHDATA
      PARAMETER (CB = 1)

      IF (MSGID .EQ. CB) THEN
99       CALL PLLOCK(SHDATA(2))
         IF (SHDATA(1) .NE. 0) THEN
            CALL PLFREE(SHDATA(2))
            GOTO 99
         ELSE
            SHDATA(0) = LOCAL_CBI
            LOCAL_CBI = LOCAL_CBI + N
            SHDATA(1) = P
         ENDIF
         CALL PLFREE(SHDATA(2))
      END IF
      RETURN
      END
```

```
      SUBROUTINE DREC(M, N, A, LDA, B, LDB, ME, PROC, CBI, MSGID)
      INTEGER CBI, MSGID, ME, M, N, LDA, PROC,LDB
      DOUBLE PRECISION A(LDA, *),B(LDB,*)
*
*     SETS CBI TO GLOBAL CURRENT BLOCK POINTER (SHDATA(0)) AND
*     DECREMENTS SHDATA(1) BY ONE TO SHOW THAT THE MESSAGE IS RECEIVED.
*
      INTEGER SHDATA(0:2), CB
      COMMON /SHARED1/ SHDATA
      PARAMETER (CB = 1)

      IF (MSGID .EQ. CB) THEN
99       CALL PLLOCK(SHDATA(2))
         IF (SHDATA(0) .EQ. CBI) THEN
            CALL PLFREE(SHDATA(2))
            GOTO 99
         ELSE
            CBI = SHDATA(0)
            SHDATA(1) = SHDATA(1) - 1
         ENDIF
         CALL  PLFREE(SHDATA(2))
      ENDIF
      RETURN
      END
```

```
      SUBROUTINE INITMESSPASS()
      INTEGER SHDATA(0:2)
      COMMON /SHARED1/ SHDATA

      SHDATA(0) = 0
      SHDATA(1) = 0
      CALL PLORIG(SHDATA(2))
      RETURN
      END
```

```
      SUBROUTINE TERMMESSPASS
      INTEGER SHDATA(0:2)
      COMMON /SHARED1/ SHDATA

      CALL PLTERM(SHDATA(2))
      RETURN
      END
```